Proceedings of the AAAI Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)

Edited and organized by

Kyle Hollins Wray
Julie A. Shah
Peter Stone
Stefan J. Witwicki
Shlomo Zilberstein

November 1, 2018

Contents

Acknowledgements

The AAAI Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA) is sponsored by *Nissan Research Center - Silicon Valley* and held in cooperation with the *Association for the Advancement of Artificial Intelligence (AAAI)* as part of the AAAI 2018 Fall Symposium Series.

Preface

Over the past decade, decision-making agents have been increasingly deployed in industrial settings, consumer products, healthcare, education, and entertainment. The development of drone delivery services, virtual assistants, and autonomous vehicles have highlighted numerous challenges surrounding the operation of autonomous systems in unstructured environments. This includes mechanisms to support autonomous operations over extended periods of time, techniques that facilitate the use of human assistance in learning and decision-making, learning to reduce the reliance on humans over time, addressing the practical scalability of existing methods, relaxing unrealistic assumptions, and alleviating safety concerns about deploying these systems.

The goal of this symposium on long-term autonomy (LTA) was to identify the challenges and bridge the gaps between theoretical frameworks for planning and learning in autonomous agents and the requirements imposed by deployment in the real world. The symposium consisted of eighteen paper presentations and three invited talks, concluding in a lively and interactive panel discussion.

In total, there were twelve long papers and six short papers, ranging in topics from planning and learning to architectures and real-world systems. The three invited talks presented work with fully operational deployments of assistant service robots, semi-autonomous vehicles, and large-scale multi-agent warehouse robots. Specifically, the techniques leveraged: hierarchical and multi-objective (PO)MDP models; reinforcement learning with general value functions; deep learning for grasping, environment understanding, and risk-aware planning; multiagent models for system robustness; and robotic architectures with a focus on tight component integration. Applications included: autonomous vehicles, delivery robots, activity recognition in smart homes, mobile warehouse robots, air traffic surveillance, dual-arm grasping robots, and mobile home healthcare robots.

Throughout the symposium four key themes emerged as topics for long-term autonomy research: (1) integration of multiple AI components beyond traditional architectural, hierarchical, and multi-objective approaches; (2) methods to proactively leverage humans to overcome any exceptional issues encountered, diminishing this reliance over time; (3) standard metrics and verification methods in order to properly measure long-term autonomous agents, such as by their exceptional issues encountered, number of human help requests, effect of system improvements made, and degree of learning performed; and (4) a focus on the robustness of the system to enable these long-term deployments.

We would like to thank everyone who submitted their papers to the LTA symposium. We would also like to thank the program committee members for their high quality reviews and feedback during the paper submission process. Additionally, we would like to thank the invited speakers for their excellent talks and collaborative panel discussion that grounded and motivated the symposium overall. Finally, we would also like to thank all of the organizing committee members as their continued effort made the LTA symposium a success.

– Kyle Hollins Wray
LTA Symposium Chair

– Shlomo Zilberstein
LTA Symposium Organizing Committee Member

Invited Speakers

Dr. Nick Hawes, Associate Professor of Robotics at the University of Oxford

Title: Learning From Four Years of Mobile Autonomy

Abstract: In this talk I will look back over four years of long-term deployments of autonomous mobile robots in everyday environments. From this I will present examples of the kinds of things that mobile robots can learn over long autonomous operations in such environments, including navigation information, human activities, object models, and mission schedules. Following this I will explore the issues (software, hardware, and social) that impacted upon the autonomy of our deployed robots, and look at what we can learn from these experiences as both AI practitioners and as engineers deploying robots in real environments.

Dr. Maarten Sierhuis, Chief Technology Director at Nissan Research Center - Silicon Valley, Founder of Ejenta

Title: Seamless Autonomous Mobility (SAM)

Abstract: Artificial intelligence will make vehicles able to drive autonomously in a wide variety of scenarios. However, unexpected situations can still arise as these long-term autonomous vehicles interact in the world, potentially limiting the uses of fully autonomous driving in the near future. Nissan's Seamless Autonomous Mobility provides a solution that can overcome this issue through the intelligent integration of humans.

Dr. Peter Wurman, VP of Engineering at Cogitai, Former Co-founder of Kiva Systems

Title: The Disruptive Power of Robots

Abstract: Kiva Systems introduced swarms of agile robots into an industry dominated by stationary conveyor systems. The path from concept through successful startup and eventual acquisition involved challenges on all fronts. In this talk I'll explain the business problem that motivated the innovation, Kiva technology and the benefits it brought to customers, and the future of applications of robotics in warehouses.

Organizing Committee

Kyle Hollins Wray, Chair
University of Massachusetts Amherst

Julie A. Shah
Massachusetts Institute of Technology

Peter Stone
University of Texas at Austin

Stefan J. Witwicki
Nissan Research Center - Silicon Valley

Shlomo Zilberstein
University of Massachusetts Amherst

Program Committee

Joydeep Biswas
University of Massachusetts Amherst

Jeremy Frank
NASA Ames Research Center

Nick Hawes
University of Oxford

David Hsu
National University of Singapore

Erez Karpas
Technion - Israel Institute of Technology

Mykel Kochenderfer
Stanford University

Sven Koenig
University of Southern California

George Konidaris
Brown University

Abdel-Illah Mouaddib
University of Caen Normandy

Nicholas Roy
Massachusetts Institute of Technology

Reid Simmons
Carnegie Mellon University

Matthijs Spaan
Delft University of Technology

Siddharth Srivastava
Arizona State University

Kiri Wagstaff
NASA Jet Propulsion Laboratory

Shiqi Zhang
The State University of New York at Binghamton

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Deep CNN and Probabilistic DL Reasoning for Contextual Affordances

Hazem Abdelkawy[*], **Sandro Rama Fiorini**[*], **Abdelghani Chibani, Naouel Ayari and Yacine Amirat**
LISSI Laboratory
University of Paris-Est Créteil (UPEC),
Vitry-sur-Seine France
{hazem-khaled-mohamed.abdelkawy, sandro.fiorini, abdelghani.chibani, amirat}@u-pec.fr

Abstract

Endowing robots with cognitive capabilities for recognising contextual object affordances is a big challenge, which requires sophisticated and novel approaches. In this paper, we propose a hybrid approach to interpret contextualised object affordances from sensor data. The proposed approach combines both Deep CNN networks for object and indoor place recognition with probabilistic DL reasoning for affordance inference. We argue that our hybrid approach can be an interesting alternative in situations where no specific dataset for contextualised affordances exists.

Introduction

Visual intelligence is one of the most important aspects of human cognition, and the paramount goal of the visual intelligence is the contextual visual reasoning. Take a cup as an example. From a single image, humans can infer its name, texture, colour, and what actions the object affords. In (Gibson 2014), Gibson defined the notion of object affordance as the "properties of an object that determine what actions a human can perform on them." In this paper, we define the notion of *contextual object affordance* as the *relationship between an object and a set of actions this object allows in a given situation*. In other words, objects might afford different actions at different places, times, or situations. In this work, contextual object affordances are proposed as means to filter the possible actions that a companion robot can monitor/do in an ambient environment. Besides, contextual affordances can be used as part of a bigger process to extract an agent intentions, by restricting the possible intentions based on the affordable actions in the environment in a given time.

The previous attempts to recognise object affordances can be divided into two categories: visual features classifications models, knowledge-based inference models. In (Fergus et al. 2005), the proposed approach is able to learn an object category from its name, based on the output of Google Image search results. In (Kjellström, Romero, and Kragić 2011), the inference of the object affordances is based on monitoring humans while they use objects in different actions. In (Yao, Ma, and Fei-Fei 2013), the proposed approach is able to model the affordance of an object based on the majority of human poses while interacting with that object. In (Chu and Thomaz 2017), object affordances are discovered by a guided exploration approach that combines self-learning with supervised learning. In (Do et al. 2017), AffordanceNet deep learning model is proposed to detect ambient objects and their affordances simultaneously from RGB images. In (Zhu, Fathi, and Fei-Fei 2014), the authors propose a Markov Logic Network (MLN) knowledge base to apply zero-shot object affordance prediction besides object recognition given human poses. Despite the previous serious attempts, only the latter model is able to predict the object affordances for unseen novel objects.

From a pure machine learning perspective, while it would be possible to train a model to produce contextual affordances, frequently data sets are not available for this task. To the best of our knowledge, no data set with these characteristics exists. Therefore alternative solutions are needed in order to use this kind of information in autonomous systems.

In this work, we propose our initial findings to extend a previously proposed cognitive architecture (Ayari et al. 2015; 2017) to predict the contextual object affordances based on place information. The extension is based on Deep Convolutional Networks (CNNs) and Probabilistic Description Logics (DL) Reasoning. The role of the probabilistic DL reasoning is to provide the ability to produce contextual affordances based on low-level object and place information. Our contribution is methodological: we demonstrate how the integration of Deep CNNs models and DL reasoning components can produce more valuable output even in a situation where the data of training is missing.

Cognitive Architecture

The overall cognitive architecture proposed in (Ayari et al. 2015; 2017) is depicted in Fig 1. At the low level, a *communication service* is implemented to enable the entities populating the ambient environment to connect and subscribe to cloud services as well as to interchange knowledge. The communication service is based on standard communication technologies such as (XMPP, REST, etc.) In addition to the communication service, emotion recognition, metric maps and topological maps based environment modelling, and multi-modal data sensing services are implemented at the low level.

[*]These authors contributed equally to this work

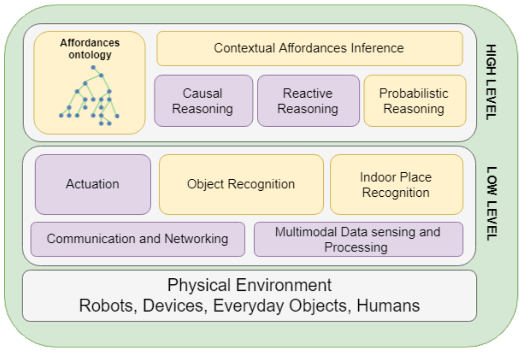


Figure 1: Overview of the architecture.

Here we focus on the *object recognition* and *indoor place recognition* services, as well as the *probabilistic reasoning service*. Their objective is to enable the companion robots to recognise the indoor places and the objects populating the environment. Implemented as Deep CNNs, models, they extract and recognise ambient objects and the indoor places in which these objects are located. This information is fed into the High Level, where the DL reasoning is able to infer the contextual object affordances based on probabilistic object and place data provided by the Low Level, using mainly instance classification and subsumption checking.

Object Recognition

We employ $YOLO$ ("$You\ Only\ Look\ Once$") algorithm (Redmon et al. 2016) in order to recognise the objects populating an environment. $YOLO$ object recognition model predicts the object bounding boxes and the associated probabilities for these boxes. Firstly, the input image is divided into SxS regions, within each region the model outputs a set of N bounding boxes. For each bounding box, the model predicts the object class probability and the location values for the bounding box. Finally, the model filters out bounding boxes with class probabilities below a predefined threshold value.

$YOLO$ model has 24 stacked convolution layers followed by 2 dense fully connected layers. To reduce the output features space, we applied 1X1 convolution reduction layers followed by 3X3 convolution layers.

Indoor Place Recognition

The Deep Residual Network $(ResNet)$ was developed by Microsoft Research Labs and exploited in (He et al. 2016) for image recognition. In this paper, we use a modified ResNet deep architecture to recognise the indoor locations. The model consists of number of stacked convolution layers, combined with residual shortcut connections to train deeper and more sparse networks. The input convolution layer consists of 64 feature maps of size 7x7 with stride of 2. A max-pooling layer of 3x3 kernel and stride of 2 is applied after the input layer to down-sample the feature maps representation. The max-pooling layer is followed by a set of 16 residual blocks, each residual block consisting of 4 convolution layers with 3x3 kernel size. A Global Average Pooling (GAP) layer (Lin, Chen, and Yan 2013) is used to minimise overfitting by reducing the total number of learned parameters. Finally, a dense, fully connected neural network with 2000 neuron is exploited as a classification layer to recognise the indoor location.

Probabilistic Reasoning

The objective of the probabilistic reasoning service is to infer object affordances based on object and place data extracted by the CNN services described above. It is based on a probabilistic DL reasoning model presented by (Riguzzi et al. 2015), supported by an OWL 2 ontology.

The ontology is relatively simple (Fig. 3). It specifies the notion of affordance as a relationship between an object instance and a type of (or class of) action. It defines three main high-level concepts: *object*, *place* and *action type*. Objects are the common objects of daily living and places are the physical places wherein these objects can be found. Objects and places are linked by the relation *placed at*. Action types are reified action concepts; its instances are types of actions which objects may afford. The reification allows one to represent affordances as a first-order relations, without requiring metamodeling subterfuges. The object relationship *affords* captures this relation. Also, representing affordances as relationships instance of class instances simplifies the reasoning, as it avoids the need of creating artificial class instances during DL reasoning, which is not trivial to control. The remainder of the ontology is a taxonomy of objects and places which parametrizes the reasoning algorithm, as well as a set of reified action types represented as instances.

The inference rules are captured by DL subsumption rules with the general formula schema

$$O \sqcap \exists \mathrm{placedAt}.P \sqsubseteq \exists \mathrm{affords}.\{T\},$$

where O is a subclass of the *Object*, P is a subclass of *Place* and T is an instance of *Action Type*. So, if any object instance of a certain class is placed at the right place, it is possible to reason that it affords a given action. Objects and places can be described as specific as necessary.

The probabilistic DL reasoning is based on DISPONTE distribution semantics (Riguzzi et al. 2015) for DL knowledge bases (KBs). In this model, DL axioms are annotated with probabilities, which are assumed to be independent. DISPONTE defines *worlds* that select subsets of axioms of the KB. The probability of a world w is defined as a joint probability over selected and non-selected axioms. Finally, the probability of a query axiom (i.e. a inferred axiom) is ultimately given by a marginalised joint probability over the worlds that entail the query. We use the BUNDLE reasoner (Riguzzi et al. 2015) carry out the inferences. BUNDLE implements DISPONTE by joining traditional DL reasoners (i.e. Pellet[1]) with Binary Decision Diagrams (BDD). In brief, given a probabilistic KB and a query axiom, BUNDLE takes worlds to be possible explanations of a query generated by a standard DL reasoner. The probability of a query then defined by marginalising over the joint probabilities of each of its explanations. This computation is optimised by

[1]https://github.com/stardog-union/pellet

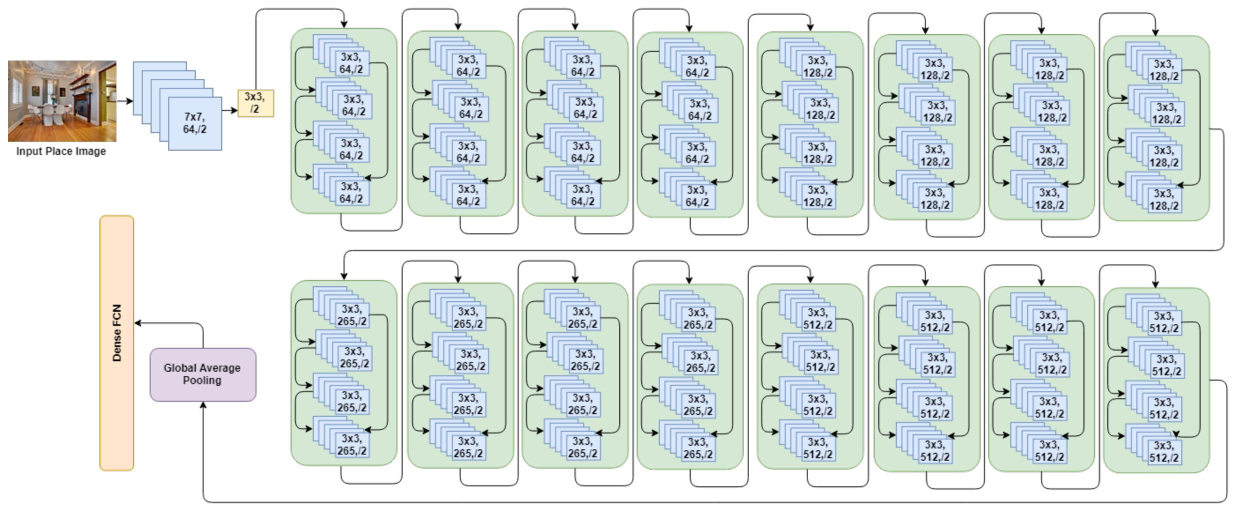


Figure 2: Deep ResNet model for indoor place recognition.

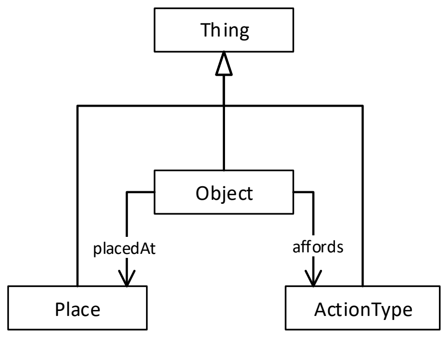


Figure 3: Overview of the ontology being used.

calculating BDDs of a disjunction of the explanations, and subsequently recursively calculating the probability of the query using the BDD.

Our affordances interpretation algorithm takes probability vectors of object and places classes detected by the CNNs as inputs. For a given input frame, the dCNN layer outputs a vector $O = [p_1 : C_1, ..., p_n : C_n]$, where C_i corresponds to the calculated class of the i-th detected object with probability p_i. For places, the dCNN layer outputs a vector $L = [p_1 : C_1, ..., p_n : C_m]$, where C_i corresponds to a possible classification for the *current* place with probability p_i. The affordances reasoner encodes this output as a set of instances of the ontology. An instance l is created in the KB and for each element $p_i : C_i$ of L, an axiom $p_i :: l : C_i$ is added to the ontology. Also, for each element $p_i : C_i$ of O, an instance x_i is created and an axiom $p_i :: x_i : C_i$ is added to the KB. All object instances x_i are defined to be *placed at* l. This KB is loaded into BUNDLE, which infers affordance relationships between instances x_i and actions types present in the model, weighted by probabilities calculated on x_i class and placement. The output of the component is the set of action types afforded by all objects in that frame, weighed by their infered probability. In cases where two objects in the same frame afford the same action, then the action type with maximum probability is taken. The object and place instances are not kept from a frame to the next.

An advantage of this method is that all the ontology axioms are taken into consideration while reasoning, even non probabilistic ones. For example, it is possible to aggregate common object classes in the object dataset under superclasses, to which one can define a single reasoning rule. Such modelling can drastically reduce the amount of inference rules to cover all possible affordances.

Preliminary Evaluation

We carried out a preliminary evaluation of the proposed method through an empirical experiment on real-world datasets. The datasets are as follows:

- Microsoft COCO (Lin et al. 2014) (Common Objects in Context) dataset was exploited to evaluate the performance of $YOLO$ object recognition deep learning model. The dataset consists of 80 different objects with total number of 2.5 million annotated instances. The dataset is divided into 118K images for training, 5K images for validation, 41K images for testing.
- The Place365 standard dataset (Zhou et al. 2017) was used to evaluate the Place recognition deep learning model. The dataset consists of 2 million images of different 365 common places. The dataset is divided into overlapped training, validation, and testing sets (1M, 36K, 300K images, respectively). The training set contains up to 5,000 images per category, while the validation and testing contains 100 and 900 images per category, respectively.
- The Daily Living Activities (ADL) dataset (Pirsiavash and Ramanan 2012) was exploited to evaluate the performance of the DL reasoning. The dataset consists of one million RGB frames of 20 persons while practising un-

scripted 32 daily activities. The dataset annotation consists of objects, activities, hand positions, and environmental events. Compared to the traditional datasets for daily activities, this dataset combining long scale temporal activities for periods up to few minutes and complex object interactions.

- The proposed ontology was populated with objects, places and related action classes from COCO, Place365, and ADL datasets respectively. We defined an initial collection of 16 DL rules to cover a subset of objects, places and action types. These rules have been defined by hand, trying to match activities in ADL to possible combinations of objects and places from COCO and Place365. By aggregating dataset objects and places into superclasses, we were able to define rules with higher reuse potential.

To evaluated the proposed approach, a set of 4577 continuous frames from the ADL dataset were used to obtain some preliminary performance statistics. In the Low Level, the Deep CNNs were able to recognise in average up to three objects ($\bar{x} = 3.43, s = 1.91$) and five indoor places (fixed value). The average processing time of recognising the ambient objects with indoor Places is 260 ms per frame of 288x384 pixels[2]. In the High Level, the reasoner component is able to generate approximately 11 axioms ($\bar{x} = 11.87, s = 3.82$), which were added to the ontology at each frame. Based on this input, the reasoner produced around 3 contextual affordances per frame in average ($\bar{x} = 3.32, s = 2.2$). The average reasoning time for recognition of contextual affordances is approximately 350 ms ($\bar{x} = 354.60, s = 216.29$) for each frame[3].

Conclusion

In this paper, we propose a hybrid approach based on Deep CNNs and DL reasoning to recognise contextual affordances. We evaluated the proposed approach through empirical experiments on real-world datasets. The preliminary evaluation shows that the processing time of the proposed approach is reasonably fitting the constraints of real-time applications.

References

Ayari, N.; Chibani, A.; Amirat, Y.; and Matson, E. T. 2015. A novel approach based on commonsense knowledge representation and reasoning in open world for intelligent ambient assisted living services. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 6007–6013.

Ayari, N.; Abdelkawy, H.; Chibani, A.; and Amirat, Y. 2017. Towards semantic multimodal emotion recognition for enhancing assistive services in ubiquitous robotics. In *2017 AAAI Fall Symposium Series*.

Chu, V., and Thomaz, A. L. 2017. Analyzing differences between teachers when learning object affordances via guided exploration. *The International Journal of Robotics Research* 36(5-7):739–758.

Do, T.; Nguyen, A.; Reid, I. D.; Caldwell, D. G.; and Tsagarakis, N. G. 2017. Affordancenet: An end-to-end deep learning approach for object affordance detection. *CoRR* abs/1709.07326.

Fergus, R.; Fei-Fei, L.; Perona, P.; and Zisserman, A. 2005. Learning object categories from google's image search. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, 1816–1823. IEEE.

Gibson, J. J. 2014. *The ecological approach to visual perception: classic edition*. Psychology Press.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Kjellström, H.; Romero, J.; and Kragić, D. 2011. Visual object-action recognition: Inferring object affordances from human demonstration. *Computer Vision and Image Understanding* 115(1):81–90.

Lin, T.; Maire, M.; Belongie, S. J.; Bourdev, L. D.; Girshick, R. B.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft COCO: common objects in context. *CoRR* abs/1405.0312.

Lin, M.; Chen, Q.; and Yan, S. 2013. Network in network. *arXiv preprint arXiv:1312.4400*.

Pirsiavash, H., and Ramanan, D. 2012. Detecting activities of daily living in first-person camera views. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2847–2854. IEEE.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Riguzzi, F.; Bellodi, E.; Lamma, E.; and Zese, R. 2015. Reasoning with probabilistic ontologies. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 4310–4316. AAAI Press.

Yao, B.; Ma, J.; and Fei-Fei, L. 2013. Discovering object functionality. In *Proceedings of the IEEE International Conference on Computer Vision*, 2512–2519.

Zhou, B.; Lapedriza, A.; Khosla, A.; Oliva, A.; and Torralba, A. 2017. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zhu, Y.; Fathi, A.; and Fei-Fei, L. 2014. Reasoning about object affordances in a knowledge base representation. In *European conference on computer vision*, 408–424. Springer.

[2]Deep CNNs running on a Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz (8 CPUs), 2.7GHz, with 16Gb RAM.

[3]Reasoner running on a Intel(R) Xeon(R) CPU E5-1630 v4 @ 3.70GHz, 4 Core(s), 8 Logical Processor(s), with 8Gb RAM.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Towards Robust Grasps: Using the Environment Semantics for Robotic Object Affordances

Paola Ardón and **Èric Pairet** and **Subramanian Ramamoorthy** and **Katrin Solveig Lohan**
Edinburgh Centre for Robotics
Heriot-Watt University and University of Edinburgh, UK.
{paola.ardon,eric.pairet,s.ramamoorthy}@ed.ac.uk;k.lohan@hw.ac.uk[*]

Abstract

Artificial Intelligence is essential to achieve a reliable human-robot interaction, especially when it comes to manipulation tasks. Most of the state-of-the-art literature explores robotics grasping methods by focusing on the target object or the robot's morphology, without including the environment. When it comes to human cognitive development approaches, these physical qualities are not only inferred from the object, but also from the semantic characteristics of the surroundings. The same analogy can be used in robotic affordances for improving objects grasps, where the perceived physical qualities of the objects give valuable information about the possible manipulation actions. This work proposes a framework able to reason on the object affordances and grasping regions. Each calculated grasping area is the result of a sequence of concrete ranked decisions based on the inference of different highly related attributes. The results show that the system is able to infer on good grasping areas depending on its affordance without having any *a-priori* knowledge on the shape nor the grasping points.

INTRODUCTION

Humanoid robots are playing increasingly important roles when it comes to indoor applications. Consider a robot assisting humans by finding, collecting and delivering an object. In such complex and dynamic environments, it is hard to provide the system with every possible representation of objects. This limitation can confuse the system into reaching very similar objects with completely different purposes, such as a candle for a glass full of liquid. Thus, the importance of a rich common sense library on object affordances that holds the start of robust robotics grasps.

Affordance is defined as "an opportunity for action" (Greeno 1994). Thus the interest in robotics on objects affordances and in artificial intelligence to investigate the best procedure to imitate the cognitive human development on how to interact with objects (Horton, Chakraborty, and Amant 2012). There is a wide range of theories that try to explain the human thinking, none of them taken as the ground truth one.

[*]The authors would like to acknowledge the support of the EPSRC IAA 455791 along with ORCA Hub EPSRC (EP/R026173/1, 2017-2021) and consortium partners.

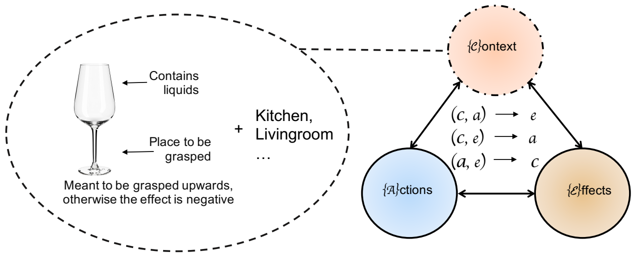


Figure 1: Affordances map model to create a correlation between the objects properties and their environment to improve on robotic grasps.

Thus, it is not surprising that the development of artificial intelligence is still a wide area of research. Humans heavily rely on shapes and environments to identify and categorise objects in order to infer an action (de Beeck, Torfs, and Wagemans 2008; Oztop, Bradley, and Arbib 2004). As a result, humans succeed at generalising an action towards objects of the same category with significantly different shapes, e.g. glasses: wine, tumbler, martini, etc., and differentiate how to manipulate objects with similar shapes but for different purposes, e.g. bowling pin vs water bottle.

In robotics, the most common approach to affordances is to learn direct mappings to labels (Bonaiuto and Arbib 2015; Hermans, Rehg, and Bobick 2011; Lenz, Lee, and Saxena 2015; Montesano et al. 2008). However, this mapping accuracy is constrained by the amount of data needed to learn the grasping areas in each of the affordance groups. These learning methods do not reveal *what are the features that encode the good object affordances?* Namely, these affordances do not strictly belong to the object itself. Instead, they are the result of the relationship established between them and the surroundings. Moreover, to engage in an interaction with humans, the robot has to be able to represent and reason with different sources of knowledge and decrease the already eminent uncertainty in the environment (Pairet et al. 2018b).

Studies on the development of human cognitive methods demonstrate that humans improve the interactive learning process with objects not only based on previous experience with them (or similar ones) but also by inferring in the context of the environment where these objects reside (Wertsh and Tulviste 1990). As a result, creating a relationship between the object, the scenario where it is more likely to be found, and the set of possible actions to interact with. Using the same analogy, in robotics, obtaining the grasp actions depending on the object affordances can be improved by integrating semantic attributes of the object and the environment in which these objects are usually found.

This paper summarises an architecture to address the challenges previously described. The presented solution builds upon the assumption that, the robot visual feedback represents a good source of information. Thus, the focus on the improvement of affordances reasoning and actions. The work establishes its foundations on the affordances map presented in Figure 1 (Montesano et al. 2008), particularly on its *context* element where the affordance identification resides. In this work the context $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ is modified to be the set of semantic attributes of the object and the environment that build upon the affordance; while $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ the set of available actions and $\mathcal{E} = \{e_1, e_2, ..., e_n\}$ the effects of performing those actions are kept as the original model.

The framework allows the system to model an unknown object and to reason on its affordance by correlating features from the target and its environment. This with the objective of calculating the best possible grasping region which is highly related to the object's affordance group. Each abstract grasping area is the result of a sequence of concrete ranked decisions based on the inference of different highly related attributes. The system combines object reconstruction methods based on geometrical approaches and deep learning techniques that delivers an efficient Knowledge Base (KB) for object affordances grasping behaviours useful in indoor environments.

RELATED WORK

Despite the wide range of methods for robotic grasps, this summary focuses on those that do not need *a-priori* information about the object in order to reconstruct it and methods that focus on the object affordance independently of the object grasp action.

Object Modelling for Grasping Based on Geometry

There are works that profit from superquadric modelling to then extract the possible grasps of an object using classifiers, (Goldfeder et al. 2009; Vezzani, Pattacini, and Natale 2017). (Goldfeder et al. 2009) integrates shape primitives and superquadrics, but the object representation is a multilevel superquadric tree. This tree is created using a decomposition of the initial model, which contains the shape primitives. After a pruning routine, a subspace containing a set of suitable grasps is obtained. (Vezzani, Pattacini, and Natale 2017) uses the superquadric modelling for both the object and the end-effector showing the method to be successful at computing the grasping area of the object and the desired pose of the end-effector.

Object Affordances for Grasping

Many methods extract viable grasping points on the objects, independently if the object is known or novel to the system, thus not explicitly considering the target's affordance. Examples of such works are (Ardón, Dragone, and Erden 2018; Lenz, Lee, and Saxena 2015; Zech and Piater 2016), to mention some. Others focus on learning the robot's control and dynamic models to achieve a grasp, such as (Stoytchev 2005; Bonaiuto and Arbib 2015). The latter learn grasp affordances from motor parameters to plan grasps using trial-and-error reinforcement learning. (Stoytchev 2005) follows psychology theories such as the ones presented in (Greeno 1994) to learn from exploratory behaviours the invariants in the resulting set of observations for the grasps.

There are also those who focus on the object affordances themselves without taking into account the grasping region. An example is (Moldovan et al. 2012) that implement a Bayesian network probabilistic method to learn to differentiate affordances models among two objects. Their proposed method shows good results under uncertainty.

In the vast repertoire of learning methods connecting affordances, not necessarily limited to objects, some works try to mimic the human reasoning by building a KB of actions based on tasks built upon reinforcement learning (Zhu, Fathi, and Fei-Fei 2014; Sridharan 2017). Instead, (Montesano and Lopes 2009; Kraft et al. 2009; Madry, Song, and Kragic 2012) learn the visual descriptors of the objects using classifiers, such as support vector machine (SVM), to categorise the objects and obtain the possible grasps. Others such as (Nguyen et al. 2017; Do, Nguyen, and Reid 2018) use classifiers alone to build a model using deep Convolutional Neural Networks (CNN) based on the visual objects features, resulting in a plausible generalised method given the robustness of their data.

PROPOSED SOLUTION AND SYSTEM INTEGRATION

The proposed framework is divided into two sub-stages as shown in Figure 2. This method focuses on modelling the object and extracting the valuable features of the target and its surrounding environment. These sets of features allow the system to deduce the target's affordance to improve the grasping actions.

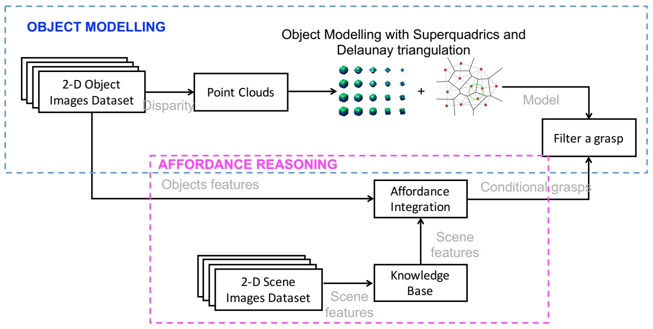


Figure 2: (a) Proposed solution to a grasping framework using affordances theory, where the context consists not only of the object but also of the environment features.

Object Modelling

Learning techniques are part of the state-of-the-art when it comes to extracting the grasping points on objects. However, they bring some limitations, such as to collect or find a suitable dataset that maps the two-dimensional (2-D) images to the labelled three-dimensional (3-D) grasping points. This approach models the object using a combination of superquadric modelling and Delaunay triangulation allowing the system to grasp novel objects without any *a-priori* information. Superquadrics are a family of geometric shapes similarly defined as ellipsoids and other quadrics, except that the squaring operations are replaced by arbitrary powers that are the ones that adapt the shape to the surface of the perceived object. The framework starts by approximating the object to a superquadric model (Jaklic, Leonardis, and Solina 2013):

$$F(x,y,z,\boldsymbol{\lambda}) : \left(\left(\frac{x}{\lambda_1}\right)^{\frac{2}{\lambda_5}} + \left(\frac{y}{\lambda_2}\right)^{\frac{2}{\lambda_5}}\right)^{\frac{\lambda_5}{\lambda_4}} + \left(\frac{z}{\lambda_3}\right)^{\frac{2}{\lambda_4}}, \tag{1}$$

where (x, y, z) is a 3-D point in the superquadric model and $\boldsymbol{\lambda} = [\lambda_1, ..., \lambda_5]$ defines the superquadric shape. Equation 1 provides a simple test whether a given point lies inside or outside a superquadric:

$$P(x,y,z) = \begin{cases} F < 1, & inside \\ F = 0, & on\ surface \\ F > 0, & outside \end{cases} \tag{2}$$

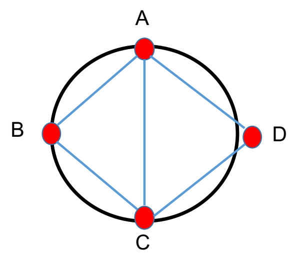
Nonetheless, one of the known problems of superquadrics is that it samples more points around the curvatures of the perceived shape (Jaklic, Leonardis, and Solina 2013).

Thus, in order to extract grasping points along the whole surface of the object, the superquadric is combined with a Delaunay triangulation. A Delaunay triangulation considers a set **P** of points in the (D-dimensional) Euclidean space. An example is shown in Figure 3. For a triangulation to be Delaunay no point in **P** should be inside the circumcircle shaped by the D-dimensional triangulation DT, with the angle vectors composed by the points in **P**, DT(**P**), formed by four chosen points inside **P** (Lee and Schachter 1980). In two dimensions, one way to detect if a point D lies in the circumcircle of A, B, C is to evaluate the determinant

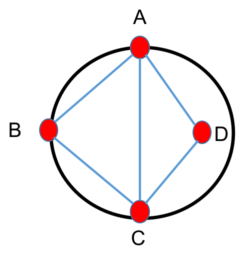
$$\begin{vmatrix} A_x & A_y & A_x^2 + A_y^2 & 1 \\ B_x & B_y & B_x^2 + B_y^2 & 1 \\ C_x & C_y & C_x^2 + C_y^2 & 1 \\ D_x & D_y & D_x^2 + D_y^2 & 1 \end{vmatrix} > 0, \tag{3}$$

where A, B and C are sorted counterclockwise. This determinant is then positive, if and only if, D is inside the cirumcircle. The vertices of the Delaunay triangulation are the ones extracted as the grasping points of the object.

Figure 4 shows the process of visualising the grasping region on the object. A superellipsoid is matched using the dimensions of iCub humanoid robot end-effector (Metta et al. 2008). This superellipsoid and the robot's hand model are portrayed in Figures 4(a) and 4(b), and an example of a modelled object with the obtained grasping region is shown in Figures 4(c) and 4(d) respectively.



(a)

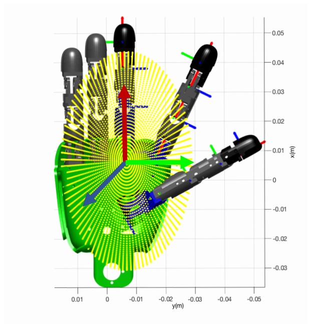


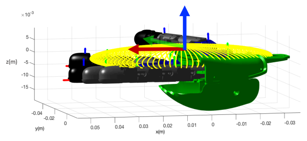
(b)

Figure 3: Delaunay Triangulation example. (a) Delaunay triangulation, (b) not a Delaunay triangulation

Building the Knowledge Base

While the previous module does not need any *a-priori* information on the object to obtain a model, reasoning about the object affordance needs a library of features that gives some background about its correct affordance group. Knowledge Base (KB) methods are growing in artificial intelligence. They learn a set of general rules and features that allow the system to infer about an object or an action. Moreover, this method is not restricted to the output task, but it also allows


(a)


(b)

(c)

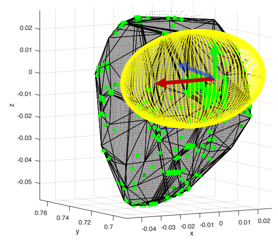

(d)

Figure 4: End-effector and object modelling. (a)-(b) show iCub humanoid Robot end-effector CAD model with its superellipsoid in yellow (axis colors: x is red, y is green and the z is blue); (c) target object used for the sample reconstruction; (d) point cloud reconstruction using superquadrics and Delaunay triangulation, the detected grasping points are shown in green and the final location of the end-effector in yellow.

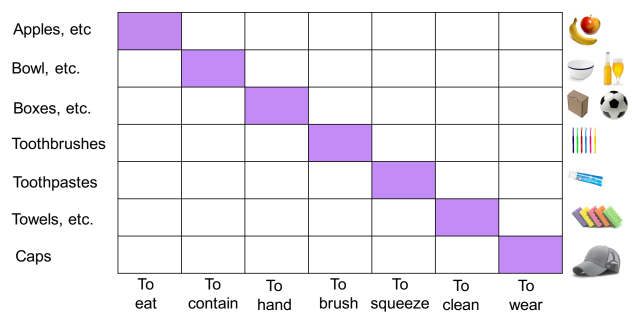


Figure 5: Objects used for our framework from the Washington dataset and the different affordances groups.

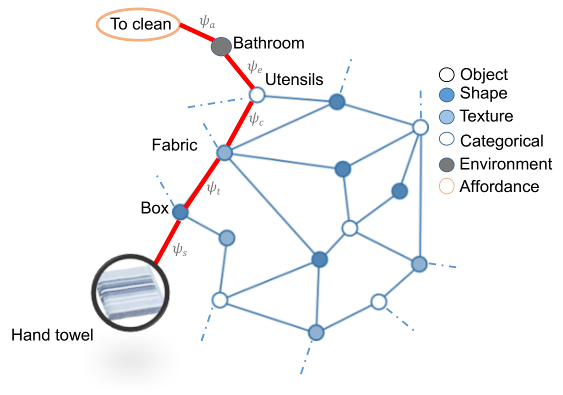


Figure 6: Example of a cleaning object and the extracted attributes used to build the KB graph learning the positive weights $\mathbf{\Psi}$ (shown in red) that result in an affordance group.

the system to query a larger array of questions regarding the features involved in the process.

In this work, a KB graph is used as a predictive model to an object affordance. The system collects a set of attributes about the objects and the environment, to then connect them in a graph style based on a set of general rules that defines the relationship among these attributes. Consequently, allowing the system to reason about the affordance group and the previously calculated grasping points. This KB consists of two steps: collecting data and learning this data relationship to reason on the affordance for grasping.

Collecting data: This is the repository of images collected from two datasets that are finally organised in the affordance categories shown in Figure 5. The first one is the Washington-RGB dataset that contains 300 objects and 51 different classes, providing the point clouds and the 2-D images for each one of the instances (Lai et al. 2012). The second dataset is the MIT Indoor scene recognition that contains 15620 different images of 67 different indoor environments (Quattoni and Torralba 2009).

Both datasets are split into 70% for training and the remaining 30% for testing. These subsets are used to train and test a battery of classifiers that help with defining good object affordances features.

Learning the knowledge base using the environment: A KB is visualised as a graph representation as illustrated in Figure 6 where the entities (nodes) are connected by general rules (edges). In this proposed solution, the entities include the target object, the object attributes and the resulting affordances groups. The general rules are the attribute to attribute relation. Weights define this relation, where the higher the weight, the higher the correlation between the two entities. The previously collected repertoire of images is used to define the attributes portrayed in Table 1 about the object:

- Shape attributes: This is defined as the set of visual attributes that describe the objects geometrical appearance,
- Texture attributes: Are a set of categories based on visual characteristics of the objects materials,
- Categorical attributes: Reflecting the semantic understanding of the object. For example, an apple is food, and
- Environment attributes: The scenarios in which the objects are more likely to be found in. This attribute is added with the purpose of facilitating the object affordances reasoning.

Figure 7 illustrates the hierarchical inference procedure followed in the KB, to arrive to an affordance group. This KB is built using four different deep learning neural networks that, through the pre-trained CNN resnet50, (He et al. 2016), extract features from the perceived images. These four different deep learning CNN correspond to the different attributes that define an entity set of the graph.

The KB is then a predictive model based on the hierarchical information obtained from the different attributes of the object (visualised as nodes in Figure 6) and the defined general rule that correlates attributes (the edges in Figure 6 from now on referred as weights). From each of the attributes, a set of weights $\Psi_{A_i} = \{\psi_1, \psi_2, ..., \psi_n\}$ is extracted hierarchically to infer on the next best entity candidate, where $\{A_i\}$ is an attribute and n the total number of entities in that attribute. The higher the ψ_n the higher the probability that the connected entities result in a better affordance inference. These weights are proportional to the posterior probability distribution obtained from the classification task. Such that the posterior probability distribution is defined as the Bayes rule:

$$\widehat{P}(a|x) = \frac{P(x|a)P(a)}{P(x)}, \tag{4}$$

Table 1: Used attributes and entities of the KB graph.

Attribute	Attribute Categories
Shape	box, cylinder, irregular, long, round
Texture	aluminium, cardboard, coarse, fabric, glass, plastic, rubber, smooth
Categorical	container, food, personal, miscellaneous, utensils
Environment	bathroom, bedroom, children room, closet, kitchen, livingroom, office

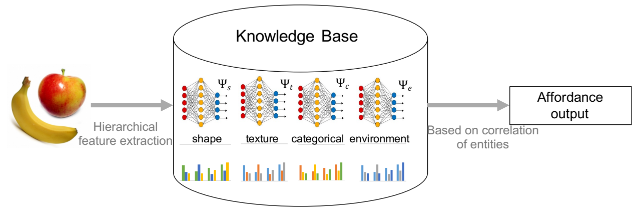


Figure 7: KB representation used for the object affordance inference. Given an image, the model estimates the attributes features in a hierarchical manner following the stated inference rule. These attributes are then accessible information on the KB. A predictive model is then applied to select the object affordance.

where x is an image belonging a class a, $P(a)$ is the posterior distribution and $P(x)$ is a normalisation constant that consists of the sum over a of the multivariate normal density. Figure 6 depicts an example of a cleaning or to hand over object, where the weights deduce the best path (shown in red) to the *to clean* affordance.

The collected information from each of the deep CNN is then learned using a decision tree as a predictive model,

$$(\boldsymbol{y}, Z) = (y_1, y_2, y_3, ..., y_n, Z), \tag{5}$$

where Z is the affordance group that the system is trying to infer and the vector $\boldsymbol{y}$ is the set of features $\{y_1, y_2, y_3, ..., y_n\}$, described as attributes categories in Table 1, used for the inference task.

Selecting the new grasping points: Once the object is classified into an affordance category, the grasping region is limited accordingly. The system selects from the set of grasping points obtained in the object reconstruction module and limits the grasps depending on the affordance action-effect of the object in the following manner:

- The grasping region should be in the middle and up-wards for objects that are meant to contain edibles.
- For the rest of objects, it is considered as the grasping region those areas where the density of grasping points is higher, given that the affordance action-effect is not critical (i.e., hand over, to brush, etc.).

SYSTEM RESULTS AND DISCUSSION

The results of the presented KB for object affordances including the environment features are presented in this section. As a reminder, the proposed framework is able to reason on the object affordance. In this work, affordance is understood as an action-effect relation of an object, with the purpose of discerning on the best possible grasps.

The current literature in affordances for grasping behaviours uses labelled grasping regions on the targets to train on the object affordance. Given that this approach aims to reason on the object grasping affordance without having any *a-priori* knowledge about its grasping regions, the presented evaluation of the results is done qualitatively.

Table 2: Each of the deep CNN accuracy performance.

Classifier	Accuracy
Shape	95.71%
Texture	98.83%
Categorical	99.91%
Environment	76.50%

Reasoning on the Object Affordance

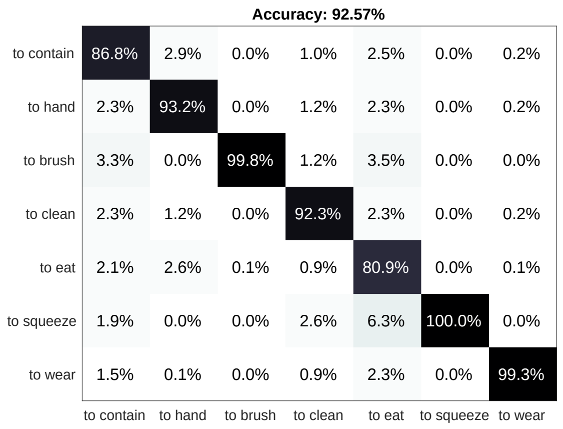
The first tests are done individually on each of the deep learning CNN that build up the KB. 30% of the images from the Washington-RGB dataset were used for testing the battery of classifiers. Table 2 presents a summary of their accuracies, whereas exhaustively presented in literature, the environment recognition is the hardest classification to boost. Even though the aim of the proposed framework is not exclusively to improve the performance of the individual classifiers, these illustrated results match the state-of-the-art results shown in (He et al. 2016; Lai et al. 2012). In order to evaluate the overall performance of the KB the accuracy and probabilities distributions before and after adding the environment features were collected.

Figures 8 and 9 show the data for both cases. Not including the environment in the affordances has lower accuracy than adding these features to the KB, as illustrated in Figures 8(a) and 8(b). Furthermore, Figure 8(a) also shows a slightly higher spread among different affordance classes. For example, the case of affordances which objects have a general semantic categorical attribute such as "miscellaneous" or "container". A percentage of objects get confused among the *to contain, to brush, to eat,* and *to squeeze* categories. Regarding grasping, this miscue represents a significant negative effect, especially for objects which real affordance is *to contain* and its misclassification results in the system ignoring the lifting-up orientation of the object, thus dropping the food or liquid inside the object. This case is reduced by 4.24% when adding the environment features, as portrayed in Figure 8(b).

The posterior probability distribution of the objects among each category is also improved. Figures 9(a) and 9(b) show the overall increase in the median probability of the objects in the different affordances categories. Further, there is a notable decrement in the distribution of categories such as *to contain, to hand, to brush*, and *to eat* meaning that the model is more confident about the classification.

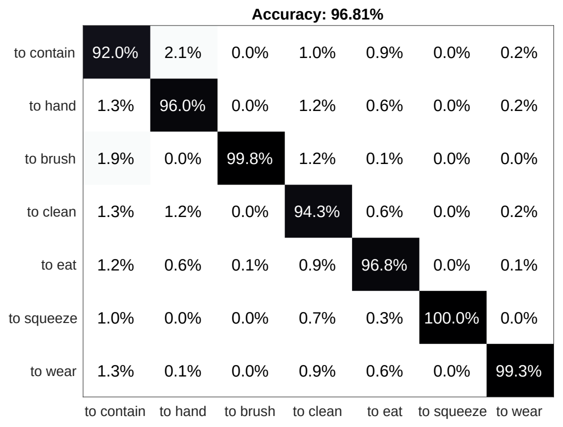
Obtained Grasping Points

The final goal is to obtain a system that, without any *a-priori* knowledge about the grasping regions of the objects, is able to reason on the affordance category and calculate the best possible grasping region. Figure 10 shows examples of different objects from which the grasping areas were extracted, before and after, inferring on the affordance of the target. These grasp regions are analysed qualitatively according to


Accuracy: 92.57%

	to contain	to hand	to brush	to clean	to eat	to squeeze	to wear
to contain	86.8%	2.9%	0.0%	1.0%	2.5%	0.0%	0.2%
to hand	2.3%	93.2%	0.0%	1.2%	2.3%	0.0%	0.2%
to brush	3.3%	0.0%	99.8%	1.2%	3.5%	0.0%	0.0%
to clean	2.3%	1.2%	0.0%	92.3%	2.3%	0.0%	0.2%
to eat	2.1%	2.6%	0.1%	0.9%	80.9%	0.0%	0.1%
to squeeze	1.9%	0.0%	0.0%	2.6%	6.3%	100.0%	0.0%
to wear	1.5%	0.1%	0.0%	0.9%	2.3%	0.0%	99.3%

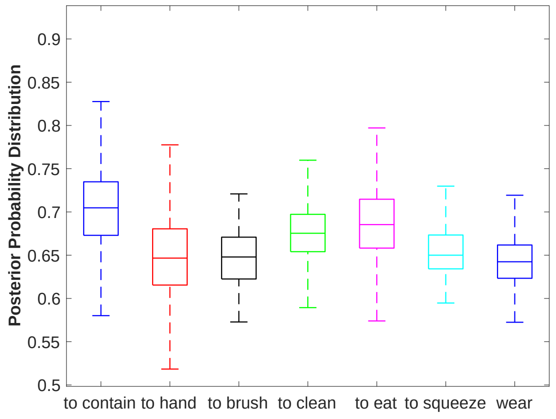
(a)


Accuracy: 96.81%

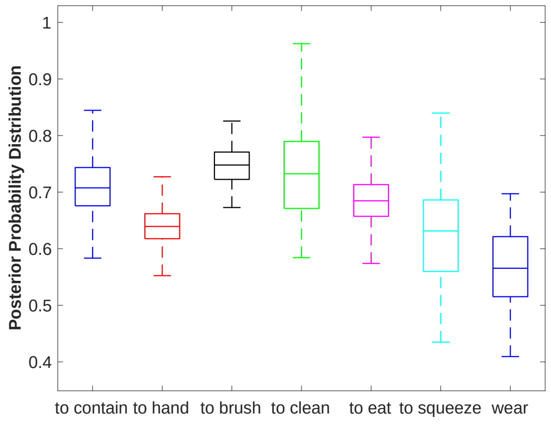
	to contain	to hand	to brush	to clean	to eat	to squeeze	to wear
to contain	92.0%	2.1%	0.0%	1.0%	0.9%	0.0%	0.2%
to hand	1.3%	96.0%	0.0%	1.2%	0.6%	0.0%	0.2%
to brush	1.9%	0.0%	99.8%	1.2%	0.1%	0.0%	0.0%
to clean	1.3%	1.2%	0.0%	94.3%	0.6%	0.0%	0.2%
to eat	1.2%	0.6%	0.1%	0.9%	96.8%	0.0%	0.1%
to squeeze	1.0%	0.0%	0.0%	0.7%	0.3%	100.0%	0.0%
to wear	1.3%	0.1%	0.0%	0.9%	0.6%	0.0%	99.3%

(b)

Figure 8: Affordance category classification performance. (a) Before adding environment features, showing an average diagonal accuracy of 92.57%; (b) After including the environment, showing an average diagonal accuracy of 96.81%.



(a)



(b)

Figure 9: Distributional posterior probabilities per class of the knowledge base. (a) Distribution before the environment inclusion, and (b) after the environment features are included.

the most likely action that a human would take in order to obtain the less negative effect.

For example, Figures 10(a) and 10(b) show the obtained model from a water bottle. In these images, the achieved grasp before deducing the affordances, results on being placed on the lid of the bottle, which would result in a negative effect if the bottle contained liquid. On the other hand, Figure 10(b) shows the calculated grasping area after the affordance has been inferred, which shows to be a more plausible solution given the risk of the object being full. The same case can be pleaded for Figures 10(c) and 10(d). In a slightly different case, Figures 10(e) and 10(f) show two different grasping regions for an object which affordance has been determined as *hand over*. Thus both grasping choices seem acceptable given that there is no critical effect involved.

FINAL REMARKS AND FUTURE WORK

Past research has presented approaches to the grasping problem extensively. However, grasping behaviours depending on the object affordances is still an open challenge due to the large variety of object shapes and robotic platforms. Furthermore, the current approaches need large amounts of data to train a model without being able to generalise among different classes of objects successfully, nor to distinguish the best grasp area depending on the object's purpose of use.

Thus, in this work, the base of a cognitive grasping framework that is able to identify and encapsulate the good affordance features of an object is presented. This task is not only limited to the relationship that can be built between the target object and the agent but also considers the surrounding environment. The results show that without any *a-priori* awareness on the grasping area of the object, the designed KB is able to induce on the object's affordance. These results are further improved by the incorporation of the environment in

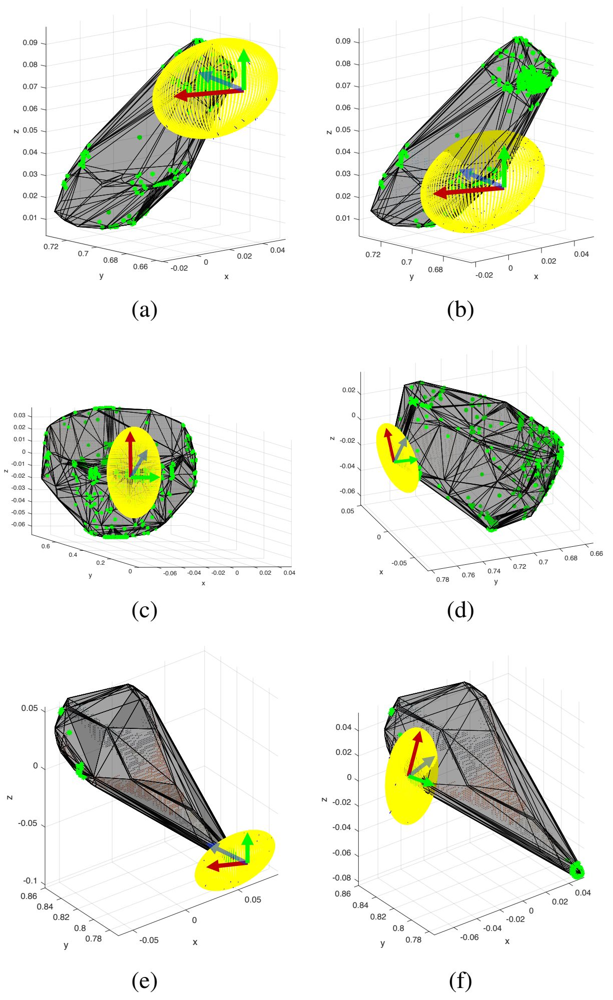


Figure 10: Objects modelling and grasping points before (left column) and after (right column) affordance reasoning, the tested objects are: first row, water bottle; second row, bowl; and third row, scissors. The extracted grasping points are shown in green while the region of the corresponding grasp is shown in yellow.

which these objects likely reside. Thus, allowing the system to have a better chance at deducing the grasping area of the object. Likewise, the presented framework has room for improvement, which is facilitated by its modularity. Overall, the performance of the KB can be increased by adding more attributes to the base, as well as modifying the predictive model in order to deal with uncertainty. Furthermore, the dynamics and system control schemes of the humanoid robot are considered out of the scope of this work. Nonetheless, (Pairet et al. 2018a) offers a learning-based framework that combines relative and absolute robotic skills for dual-arm manipulation suitable for dynamic environments tasks such as grasping objects that together with the semantics of the object offer a complete manipulation platform for humanoid robots.

References

Ardón, P.; Dragone, M.; and Erden, M. S. 2018. Reaching and grasping behaviours by humanoid robots through visual servoing. In *Haptics: Science, Technology and Applications, Springer International Publishing AG*, 353–365. Springer Nature.

Bonaiuto, J., and Arbib, M. A. 2015. Learning to grasp and extract affordances: the Integrated Learning of Grasps and Affordances (ILGA) model. *Biological cybernetics* 109(6):639–669.

de Beeck, H. P. O.; Torfs, K.; and Wagemans, J. 2008. Perceived shape similarity among unfamiliar objects and the organization of the human object vision pathway. *Journal of Neuroscience* 28(40):10111–10123.

Do, T.-T.; Nguyen, A.; and Reid, I. 2018. Affordancenet: An end-to-end deep learning approach for object affordance detection. In *International Conference on Robotics and Automation (ICRA)*.

Goldfeder, C.; Ciocarlie, M.; Peretzman, J.; Dang, H.; and Allen, P. K. 2009. Data-driven grasping with partial sensor data. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 1278–1283. IEEE.

Greeno, J. G. 1994. Gibson's affordances. *Psychological Review* 336–342.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hermans, T.; Rehg, J. M.; and Bobick, A. 2011. Affordance prediction via learned object attributes. In *IEEE International Conference on Robotics and Automation (ICRA): Workshop on Semantic Perception, Mapping, and Exploration*, 181–184. Citeseer.

Horton, T. E.; Chakraborty, A.; and Amant, R. S. 2012. Affordances for robots: a brief survey. *AVANT. Pismo Awangardy Filozoficzno-Naukowej* 2:70–84.

Jaklic, A.; Leonardis, A.; and Solina, F. 2013. *Segmentation and recovery of superquadrics*, volume 20. Springer Science & Business Media.

Kraft, D.; Detry, R.; Pugeault, N.; Baseski, E.; Piater, J. H.; and Krüger, N. 2009. Learning objects and grasp affordances through autonomous exploration. In *ICVS*.

Lai, K.; Bo, L.; Ren, X.; and Fox, D. 2012. Detection-based object labeling in 3d scenes. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 1330–1337. IEEE.

Lee, D.-T., and Schachter, B. J. 1980. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences* 9(3):219–242.

Lenz, I.; Lee, H.; and Saxena, A. 2015. Deep learning for detecting robotic grasps. *International Journal of Robotics Research* 34(4-5):705–724.

Madry, M.; Song, D.; and Kragic, D. 2012. From object categories to grasp transfer using probabilistic reasoning. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 1716–1723. IEEE.

Metta, G.; Sandini, G.; Vernon, D.; Natale, L.; and Nori, F. 2008. The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, 50–56. ACM.

Moldovan, B.; Moreno, P.; van Otterlo, M.; Santos-Victor, J.; and De Raedt, L. 2012. Learning relational affordance models for robots in multi-object manipulation tasks. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 4373–4378. IEEE.

Montesano, L., and Lopes, M. 2009. Learning grasping affordances from local visual descriptors. In *Development and Learning, 2009. ICDL 2009. IEEE 8th International Conference on*, 1–6. IEEE.

Montesano, L.; Lopes, M.; Bernardino, A.; and Santos-Victor, J. 2008. Learning object affordances: From sensory–motor coordination to imitation. *IEEE Trans. Robotics* 24:15–26.

Nguyen, A.; Kanoulas, D.; Caldwell, D. G.; and Tsagarakis, N. G. 2017. Object-based affordances detection with convolutional neural networks and dense conditional random fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Oztop, E.; Bradley, N. S.; and Arbib, M. A. 2004. Infant grasp learning: a computational model. *Experimental brain research* 158(4):480–503.

Pairet, È.; Ardón, P.; Brox, F.; Mistry, M.; and Petillot, Y. 2018a. Learning and generalisation of primitives skills towards robust dual-arm manipulation. In *AAAI Fall Symposium. Artificial Intelligence for Reasoning and Learning in Real-World Systems for Long-Term Autonomy*. AAAI Press.

Pairet, È.; Hernández, J. D.; Lahijanian, M.; and Carreras, M. 2018b. Uncertainty-based Online Mapping and Motion Planning for Marine Robotics Guidance. In *Intelligent Robots and Systems (IROS), 2018 IEEE/RSJ International Conference on*. IEEE.

Quattoni, A., and Torralba, A. 2009. Recognizing indoor scenes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 413–420. IEEE.

Sridharan, M. 2017. Integrating knowledge representation, reasoning, and learning for human-robot interaction. In *AAAI Fall Symposium. Artificial Intelligence for Human-Robot Interaction*, 69–76. AAAI Press.

Stoytchev, A. 2005. Toward learning the binding affordances of objects: A behavior-grounded approach. In *Proceedings of AAAI symposium on developmental robotics*, 17–22.

Vezzani, G.; Pattacini, U.; and Natale, L. 2017. A grasping approach based on superquadric models. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 1579–1586. IEEE.

Wertsh, J. V., and Tulviste, P. 1990. Apprenticeship in thinking: Cognitive development in social context. *Science* 249(4969):684–686.

Zech, P., and Piater, J. 2016. Active and transfer learning of grasps by sampling from demonstration.

Zhu, Y.; Fathi, A.; and Fei-Fei, L. 2014. Reasoning about object affordances in a knowledge base representation. In *European conference on computer vision*, 408–424. Springer.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

From Abstract to Executable Models for Multi-Agent Path Finding on Real Robots

Roman Barták, Jiří Švancara, Věra Škopková, David Nohejl
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic
bartak@ktiml.mff.cuni.cz

Abstract

Multi-agent path finding (MAPF) deals with the problem of finding a collision-free path for a set of agents (robots). An abstract model with a graph describing the environment and agents moving between the nodes of the graph has been proposed. This model is widely accepted by the MAPF community and majority of MAPF algorithms rely on this model. In this paper we argue that the model may not be appropriate, when the plans are to be executed on real robots. We provide some preliminary empirical evidence that abstract plans deviate from real plans executed on robots and we compare several variants of abstract models. The paper motivates further research on abstraction of problems with respect to applicability of solutions in practice.

Introduction

Abstraction is the process of removing details from a problem representation. It is a critical step in problem solving as without abstraction "intelligent agents would be completely swamped by the real world" (Russell and Norvig 2009). Despite its importance, little attention has been paid to abstraction techniques compared to, for example, solving techniques. In areas, such as planning, the formal abstract model has been proposed and many concrete domain models are used for benchmarking, but the studies how to obtain such models and how the models relate to real world are rare.

In this paper, we look at a specific planning problem called *multi-agent path finding* (MAPF) that deals with finding collision-free paths for a set of agents. We selected this problem for several reasons. First, MAPF has a strong practical applicability in areas such as warehousing and intelligent road junctions. Second, there exists a widely-accepted uniform abstract model of MAPF that uses only a few abstract types of actions that are easily executed on real robots.

Our goal is studying appropriateness of MAPF abstract models from the perspective of executing the obtained plans. We will present the core abstract model used by state-of-the-art solvers together with several extensions closer to reality. The obtained plans will be empirically compared by executing them on real robots called Ozobots (Ozobot & Evollve, Inc. 2018). This is a short version of paper (Barták et al. 2018), which gives full technical details. We focus on motivating this type of research and on discussing futures steps.

Background on MAPF

Formally, the MAPF problem is defined by a graph $G = (V, E)$ and a set of agents $a_1, \ldots, a_k$, where each agent a_i is associated with starting location $s_i \in V$ and goal location $g_i \in V$. The time is discrete and in every time step each agent can either move from its location to a neighboring location or wait in its current location. A grid map with a unit length of each edge is often used to represent the environment (Ryan 2008). The task is to find a collision-free path for each agent, where the collision occurs when two agents are at the same node at the same time or two agents move along the same edge at the same time in opposite directions. The makespan (the maximal time when all agents reached their destinations) objective function is often studied in the literature (Surynek 2014). The problem to find a makespan-optimal solution is NP-hard (Yu and LaValle 2013). Though the plans obtained by different MAPF solvers might be different, the optimal plans are frequently similar and tight (no superfluous steps are used). Hence, any optimal MAPF solver can be used. We used the reduction-based solver in the Picat programming language (Barták et al. 2017).

MAPF Models and Executable Plans

For our study we designed an environment that is intentionally close to the abstract model of MAPF, that is, it is a grid map with equal distances between vertices that are connected by lines used by robots to easily navigate between the vertices, see Figure 1. The abstract plan outputted by MAPF solvers is, as defined, a sequence of locations that the agents visit. However, a physical agent has to translate these locations to a series of actions that the agent can perform. We assume that the agent can turn left and right and move forward. By concatenating these actions, the agent can perform all the required steps from the abstract plan (recall, that we are working with grid worlds). This translates to five possible actions at each time step - (1) wait, (2) move forward, (3,4) turn left/right and move, and (5) turn back and move. As the mobile robot cannot move backward directly, turning back is implemented as two turns right (or left). Ozobot robots, used in our study, can directly perform these actions, which together with the specific map simplifies typical "robotics" problems such as localization and control.

As the abstract steps may have durations different from the physical steps, the abstract plans, which are perfectly

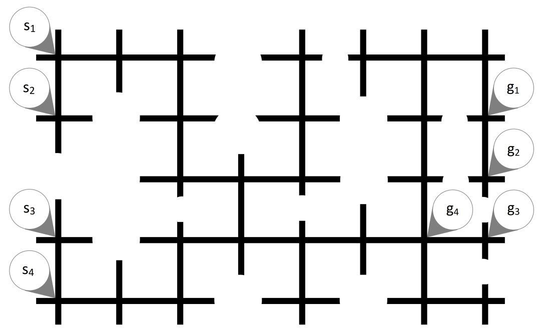


Figure 1: Instance map for Ozobots. Ozobots follow the black line, the gray circles indicate starting and goal locations (not printed on real map).

synchronized, may desynchronize when being executed, which may further lead to collisions. The intuition says that such desynchronization will indeed happen. In our setting, the speed of the robots was set in such a way that moving along a line takes 1600ms and turning takes 800ms. Note that the real robots only blindly follow the computed plan and do not intervene if, for example, an obstacle is detected.

In the rest of the section, we describe the studied abstract MAPF models and possible transformations of abstract plans to executable sequences of physical actions. Let t_t be the time needed by the robot to turn by 90 degrees to either side and t_f be the time to move forward to the neighboring vertex in the grid. Both t_t and t_f are nonzero. The time spend while the agent is performing the wait operation t_w will depend on each model.

Classical Model

The first and most straightforward model is a direct translation of the abstract plan to the action sequence. We shall call this a *classic* model. At the end of each timestep, an agent is facing in a direction. Based on the next location, the agent picks one of the five actions described above and performs it. This means that all move actions consist of possible turning and then going forward. There are no independent turning moves. As the two most common actions in abstract plans are (2) and (3,4), we suggest to set the time t_w of waiting actions to be $t_f + 1/2 * t_t$ as the average of durations of actions (2) and (3,4).

One can easily see that this simple model can be prone to desynchronization, as turning adds time over agents that just move forward. To fix this synchronization issue, we introduce a *classic+wait* model. The basic idea is that each abstract action takes the same time, which is realized by adding some wait time to "fast" actions. The longest action is (5), therefore each action now takes $2 * t_t + t_f$ including the waiting action. The consequence is that plan execution takes longer time, which may not be desirable.

Note that both of these models do not require the MAPF algorithm and model to change. They only use different durations of abstract actions which are implemented in the translation of abstract plans to executable actions.

Robust Model

Another way to fix the synchronization problem is to create a plan π that is robust to possible delays during execution. The k-robust plan is a valid MAPF plan that in addition requires for each vertex of the graph to be unoccupied for at least k time steps before another agent can enter it (Atzmon et al. 2017). In our experiments, we choose k to be 1. We presume that this is a good balance between keeping the agents from colliding with each other while not prolonging the plan too much. The 1-robust plan is then translated to executable actions using the same principle as the *classic* model. This yields a *1-robust* model. Though, this model does not solve the synchronization issue directly, it adds some slack that can prevent collisions caused by various reasons.

Split Actions Model

By making the model less abstract, we can directly represent the executable actions, in particular, by introducing an abstract turning action. In the reduction-based solvers, this can be done by splitting each vertex v_i from the original graph G into four new vertices $v_i^{up}, v_i^{right}, v_i^{down}, v_i^{left}$ indicating directions where the agent is facing to. The new edges now represent the turn actions, while the original edges correspond to move only actions. This change needs to be accompanied by constraints restricting the agents not to be at split vertices at the same time. The abstract plan is then translated to an executable plan in a direct way as the agent is given a sequence of individual actions wait, turn left/right, and move forward. The waiting time t_w is set as the bigger time of the remaining actions: $t_w = \max(t_t, t_f)$. We shall call this a *split* model.

To make the model even closer to reality, we can exploit the weighted MAPF (Barták, Švancara, and Vlk 2018), where each edge in the graph is assigned an integer value that denotes its length. The weighted MAPF solver finds a plan that takes these lengths into account. The lengths of turning edges are assigned a length of t_t and the other edges are assigned a length of t_f (or its scaled value to integers). The waiting time t_w is set as the smaller time of the remaining actions: $t_w = \min(t_t, t_f)$. We shall call this a *weighted-split* model or *w-split* for short.

A final enhancement to the *weighted-split* model is to introduce k-robustness there. This will again ensure that the agents do not tend to move close to each other to avoid undesirable collisions. In this case, however, it is not enough to use 1-robustness, as the plan is split into more time steps. Instead, we use $\max(t_t, t_f)$-robustness. We shall call this *robust-weighted-split* model or *rw-split* for short.

Results of Experiments

We generated plans using each MAPF model for the problem instance described above and then we executed the plans five times in total for each model. Several properties were measured with results shown in Table 1.

Computed makespan is the makespan of the plan returned by the MAPF solver. It is measured by the (weighted) number of abstract actions and this is the value optimized by the solvers. Note that the *split* models have larger makespan

	Comp. Mksp	Failed Runs	#Colls.	Total Time [s]	Max Δ [s]
classic	17	5	4	NA	5
classic+wait	17	0	4.2	53	0
1-robust	19	0	0	41	4
split	27	0	2	36	3
w-split	45	0	2.6	39	0
rw-split	47	0	0	39	0

Table 1: Real performance of Ozobots for studied models.

than the rest because the *split* models use a finer resolution of actions, namely turning actions are included in the makespan calculation. This is even more noticeable with *w-split* and *rw-split*, where the moving-forward action has a duration (weight) of two. Total time is the actual time needed to complete the plan by all robots. To measure the level of desynchronization, we introduced the Max Δ time. We made abstract plans for all robots equally long by adding void wait actions to the end (where necessary). The Max Δ time is the time difference between the real end times of the first and last robots. This value is zero, if the robots remained synchronized during plan execution. The larger value means larger desynchronization. All of the times are rounded to seconds because the measurements were conducted by hand.

The number of failed runs is also shown. The only model that did not finish any run is the *classic* model while the rest managed to finish all of the runs. A run fails if there is a collision that throws any of the robots off the track so the plan cannot be finished. The average number of collisions per run shows how many collisions that did not ruin the plan occurred. These collisions can range from small one, where the robots only touched each other and did not affect the execution of the plan, to big collisions, where the agent was slightly delayed in their individual plan, but still managed to finish the plan. For the *classic* model, where no execution finished, we present the number of collisions occurring before the major collision that stopped the plan.

Conclusions and Future Steps

The goal of the paper is showing that abstract models should be treated more carefully, when the results are supposed to by used in real environment. Our preliminary experiment showed that the most widely used MAPF model, the *classic* one, is actually not applicable even if the environment is made very close to the model. The reason is that durations of real actions are different from durations of abstract actions, which leads to desynchronization of agents' plans. A naive extension to make all actions equally long worsens the quality of plan (makespan) significantly. Adding robustness to abstract plans helps, but as the Max Δ time shows, there is some desynchronization, which may lead to collisions for longer plans. The *split* model uses abstraction closer to reality and adding weights makes the abstract plans even closer to real plans when executed. However, solving such models is more computationally expensive than solving the classical model (Barták, Švancara, and Vlk 2018).

The results show that there is indeed a gab between widely-used theoretical frameworks for MAPF and deployment of solutions in real environments. A wider experimental study is necessary to understand better the relations between abstract models and real environments. For example, the ratio between the length of edges and the size of robots seems important (Ozobots have diameter of 3 cm and distance between nodes in our map is 5 cm). Note also, that blind execution of plans was assumed. It would be interesting to look at plan-execution policies that assume communication between agents and exploit information from sensors (Ma, Kumar, and Koenig 2017).

Acknowledgements

Roman Barták is supported by the Czech Science Foundation under the project P202/12/G061 and together with Jiří Švancara by the Czech-Israeli Cooperative Scientific Research Project 8G15027.

References

Atzmon, D.; Felner, A.; Stern, R.; Wagner, G.; Barták, R.; and Zhou, N. 2017. k-robust multi-agent path finding. In *Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS)*, 157–158.

Barták, R.; Zhou, N.-F.; Stern, R.; Boyarski, E.; and Surynek, P. 2017. Modeling and solving the multi-agent pathfinding problem in picat. In *29th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 959–966. IEEE Computer Society.

Barták, R.; Švancara, J.; Škopková, V.; and Nohejl, D. 2018. Multi-agent path finding on real robots: First experience with ozobots. In *Advances in Artificial Intelligence – IBERAMIA 2018*. Springer.

Barták, R.; Švancara, J.; and Vlk, M. 2018. A scheduling-based approach to multi-agent path finding with weighted and capacitated arcs. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 748–756.

Ma, H.; Kumar, T. K. S.; and Koenig, S. 2017. Multi-agent path finding with delay probabilities. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 3605–3612. AAAI Press.

Ozobot & Evollve, Inc. 2018. Ozobot — Robots to code, create, and connect with.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Ryan, M. R. K. 2008. Exploiting subgraph structure in multi-robot path planning. *J. Artif. Intell. Res.* 31:497–542.

Surynek, P. 2014. Compact representations of cooperative path-finding as SAT based on matchings in bipartite graphs. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI*, 875–882. IEEE Computer Society.

Yu, J., and LaValle, S. M. 2013. Structure and intractability of optimal multi-robot path planning on graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Behavior Modeling for Autonomous Driving

Aniket Bera
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC

Dinesh Manocha
Department of Computer Science and
Electrical & Computer Engineering
University of Maryland at College Park
College Park, MD

Abstract

We present a novel approach to automatically identify driver behaviors from vehicle trajectories and use them for safe navigation of autonomous vehicles. We propose a novel set of features that can be easily extracted from car trajectories. We derive a data-driven mapping between these features and six driver behaviors using an elaborate web-based user study. We also compute a summarized score indicating a level of awareness that is needed while driving next to other vehicles. We also incorporate our algorithm into a vehicle navigation simulation system and demonstrate its benefits in terms of safer real-time navigation, while driving next to aggressive or dangerous drivers.

Introduction

There are different kinds of drivers in urban environments, and an expert human driver will identify dangerous drivers and avoid them accordingly. However, existing autonomous driving systems often treat all neighboring vehicles the same and do not take actions to avoid the dangerous drivers. This problem has been studied in transportation and urban planning works (Meiring and Myburgh 2015). This line of works map drivers' behaviors with background information like age, gender, driving history, etc., but this information is not available to autonomous vehicles. Therefore, to allow autonomous driving algorithms to account for driving behaviors, a mapping between sensor data and driving behaviors must be available.

Previous studies in transportation and urban studies (Feng et al. 2012) usually study the difference between aggressive drivers, careful drivers and typical drivers. In particular, Guy et al. (Guy et al. 2011) and Bera et al. (Bera, Randhavane, and Manocha 2017) applied psychological theory to capture human behaviors. Autonomous driving systems that are on the roads right now uses a range of different algorithms to interpret the sensor data: trajectory data computation using semantic understanding or object detection methods (Geiger, Lenz, and Urtasun 2012). Some uses an end-to-end approach to compute driving actions directly from sensor data(Bojarski et al. 2016).

Main Results: Our approach takes into account behaviors of neighboring entities and plans accordingly to perform safer navigation. We leverage the results of an extensive user study that learned the relationship between vehicular trajectories and the underlying driving behaviors: Trajectory to Driver Behavior Mapping (Cheung et al. 2018). This work allows our navigation algorithms to classify the driving behaviors of neighboring drivers, and we demonstrated simulated scenarios with vehicles, pedestrians, and cyclist where navigation with our approach is safer.

Compared to prior algorithms, our algorithm offers the following benefits:

1. Driving Behavior Computation: Trajectory to Driver Behavior Mapping established a mapping between five features and six different driving behaviors, and conducted factor analysis on the six behaviors, which are derived from two commonly studied behaviors: aggressiveness and carefulness. The results show that there exists a latent variable that can summarize these driving behaviors and that can be used to measure the level of awareness that one should have when driving next to another vehicle. The same study examined how much attention a human pays to such a vehicle when it is driving in different relative locations. We leverage the results of this study and develop a proximity cost that reacts to aggressive drivers more appropriately.

2. Improved Realtime Navigation: We enhance an existing Autonomous Driving Algorithm (Best et al. 2017) to navigate according to the neighboring drivers' behaviors. Our navigation algorithm identifies potentially dangerous drivers in real-time and chooses a path that avoids potentially dangerous drivers. In particular, our approach accounts for pedestrians and cyclists, and avoids them by considering their velocity relative to the ego-vehicle. Our method can offer saver navigation and plan more appropriately to avoid dangerous drivers than prior works. We refer the readers to read (Cheung et al.) for the technical details.

An overview of our approach is shown in Figure 1. The rest of the paper is organized as follows. We present a detailed overview of previous work in Section . We describe the mapping from trajectories to driving behaviors in Section and our autonomous driving algorithm in Section .

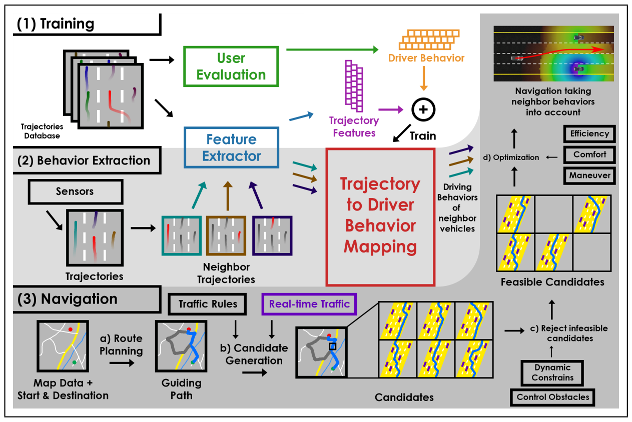


Figure 1: *Overview of our Algorithm: (1) Training: a trajectories database is training a mapping between trajectory features and driving behaviors. (2) Behavior Extraction: During navigation, the same set of features is extracted from neighboring vehicles' trajectories and mapped to driving behaviors. (3) Navigation: a) the navigation algorithm first plans a global route in accordance with map data, starting point, and destination, and b) generates a set of candidate local routes that obey traffic rules while considering real-time traffics; c) the algorithm then removes infeasible candidates using dynamic constrains and control obstacles; d) after that, it performs an optimization to obtain the best navigation plan based on the driving behavior we extracted in (2), along with several other factors: Efficiency, Passenger Comfort, etc.*

Related Works

Driving Behaviors Studies

Psychology, transportation, and urban planning researchers have been studying human driving behaviors. Aljaafreh et al. (Aljaafreh, Alshabatat, and Al-Din) classified drivers into four different levels of aggressiveness with accelerometer data. Feng et al. (Feng et al. 2012) categorized drivers into three different level of aggressiveness according to drivers' background information (age, gender, experience, etc.), and environmental factors (weather, traffic, etc.). Apart from that, social psychologist have also studied the correlation between driver background information and driving behaviors (Krahé and Fenske 2002; Beck, Ali, and Daughters 2014), and previous driving behaviors (Brill et al. 2009). Besides, Meiring et al. (Meiring and Myburgh 2015) pointed out that careless drivers, including drunk and distracted drivers, are also dangerous. Despite the fact that these works have found mappings between driving behaviors and a lot of other different factors, most of these factors are unknown to autonomous vehicles during navigation. We use neighboring vehicles' trajectories, which can be computed from sensor data, to map driving behaviors.

The following works have conducted analysis on aggressiveness and carefulness in accordance to trajectory related data. Qi et al. (Qi et al. 2015) presented the relationship between driving style, speed, and acceleration. Shi et al. (Shi et al. 2015) concluded that measuring throttle opening is better than merely measuring acceleration, as measuring deceleration (negative acceleration) is not helpful in understanding the aggressiveness of a driver. Murphey et al. (Murphey, Milton, and Kiliaris) presented results to show that measuring longitudinal jerk (changing lanes) is more helpful than progressive jerk (along the traffic direction) in terms of correlation to aggressiveness of drivers. Mohamad et al. (Mohamad, Ali, and Ismail) performed abnormal detection using speed, acceleration, and steering wheel movement. Sadigh et al. (Sadigh et al. 2014) proposed a Convex Markov Chains model to predict the attention drivers spend on driving. There are also works that are deployed in cars to sound an alert when they find the user is not paying attention to the road (Goldman 2011). Besides, there is considerable number of simulated driving models(Treiber, Hennecke, and Helbing 2000) that have proposed different factors that imply driving behaviors that can be mapped to navigation plans. Our work leverages the results from a detailed user study described in Section to use the most relevant trajectory features to driving behaviors.

Adaptation to Human Drivers' Behaviors

One line of work went further to study how humans would react to an autonomous vehicle's actions. Sadigh et al. (Sadigh et al. 2016) discovered that human drivers' behaviors can be affected when they observe an autonomous vehicle and that they will react in certain ways when they observe different actions of the autonomous vehicle (Sadigh et al.). Huang et al. (Huang et al. 2017) proposed a technique to make autonomous car actions more easily understand by humans, so that their reactions are more predictable. Besides, an active learning approach (Dorsa Sadigh, Sastry, and Seshia 2017) using examples of expert human driver's preferences has been to model human driving behaviors. These works show the importance of having autonomous vehicles navigating according to human behaviors.

Autonomous Car Navigation

There is a significant number of works on navigating autonomous vehicles (Katrakazas et al. 2015; Saifuzzaman and Zheng 2014; Ziegler et al. 2014; Kolski et al. 2006; Hoffmann et al. 2007). During the DAPRA Urban Grand Challenge and the Grand Cooperative Driving Challenge, the participating research teams proposed different navigation approaches (Buehler, Iagnemma, and Singh 2009). Recently, Best et al. (Best et al. 2017) proposed a novel navigation algorithm, AutonoVi, which also considers steering and acceleration planning, dynamic lane changes, and several other scenarios. We proposed a new approach that takes into account driving behavior, which is complimentary to these previous work and can be combined with them.

Trajectory to Driver Behavior Mapping

In this section, we describe the trajectory features that are used to identify driver behaviors, the driving behavior metrics, and the attention metrics used in a detailed user study, Trajectory to Driving Behavior Mapping (Cheung et al. 2018).

Features

The goal of Trajectory to Driving Behavior Mapping is to leverage a set of trajectory features that map to driving behaviors, assuming that the trajectories have already been extracted from the raw sensor data. As described in the previous section, a lot of features (e.g., drivers' backgrounds, throttle opening, environmental factors, etc.) that have been mapped to driving behavior are not available for autonomous vehicles. Therefore, the user study has derived a set of variants and performed feature selection to select the most relevant ones to use in the mapping.

Notation	Description
v_{nei}	Relative speed to neighbors
v_{avg}	Average velocity
s_{front}	Distance with front car
j_l	Longitudinal jerk
s_{center}	Lane following metric

Table 1: *Five Features selected in Trajectory to Driving Behavior Mapping*

Acceleration Previous works (Murphey, Milton, and Kiliaris) have shown that acceleration can be used to identify driver aggressiveness. This study (Murphey, Milton, and Kiliaris) found out that longitudinal jerk can reflect aggressiveness better than progressive jerk, and this has been further verified during the feature selection in the user study.

Lane following The metric proposed in this work (Bergasa et al. 2014) measures the extent of lane following using the mean and standard deviation of lane drifting and lane weaving. Trajectory to Driving behavior proposes a feature that also depends on lane drifting, but further differentiates drivers who keep deviating from the center of the lane to the left and right, and those drivers who are driving stably off the center of the lane. Furthermore, when a vehicle is performing lane changing, the effect on this metric of these trajectory segments is nullified and will not impact this metric.

Let y_l and $y(t)$ be the center longitudinal position of the lane in which the targeted car is in and the longitudinal position of the car at time t, respectively. Also suppose a set of lane changing events happened at time t_i, $C = \{t_1, t_2, ..., t_n\}$, the lane drift metric $s_C(t)$ is given by:

$$s_C(t) = \begin{cases} 0, & \text{if } \exists t \in C \text{ s.t. } t \in [t-k, t+k], \\ y(t) - y_l, & \text{otherwise.} \end{cases} \tag{1}$$

where k is the amount of time that we nullify the impact of lane changing to this metric.

Trajectory to Driving Behavior Mapping measures the rate of change in drifting in τ seconds, so that this metric can highlight those drivers who are drifting more frequently from the center of the lane. The overall lane following metric is therefore defined as below. It is also illustrated in Figure 2.

$$s_{center} = \int |s_C(t)| \left[\mu + \int_{t-\tau}^{t} |s'_{\emptyset}(t)| dt \right] dt, \tag{2}$$

where μ is a parameter that differentiates drivers who are driving stably off the center of the lane, and those who are driving along the center of the lane.

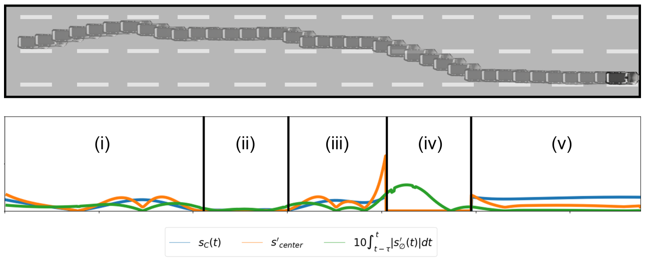


Figure 2: *Lane following metric illustration. The lane following metric, s_{center}, is given by the sum of the area under the plot s'_{center}. The example shows that the lane following metric can differentiate drivers from drifting left and right (i iii), driving along the center of the lane (ii), changing lanes (iv), and consistently driving off the center of the lane (v).*

Relative Speed Trajectory to Driving Behavior Mapping designed the following metric to capture the relationship between a given driving behavior and the relative speed of the car with respect to neighboring cars:

$$v_{nei} = \int \sum_{n \in N} \max(0, \frac{v(t) - v_n(t)}{dist(x(t), x_n(t))}) dt, \tag{3}$$

where N is the set containing all neighboring cars within a reasonably huge range. $v(t)$, $x(t)$, $v_n(t)$, $x_n(t)$ are the speed and the position of the targeting car, and the position and the speed of the neighbor n, respectively.

This metric relies merely on the speed and position of the neighbors, and it can represent the actual driving speed of the targeted vehicle with respect to it's neighbor better than simply using relative speed.

Driving Behavior Metrics and Attention Metrics

Aggressiveness (Feng et al. 2012; Aljaafreh, Alshabatat, and Al-Din ; Harris et al. 2014) and Carefulness (Meiring and Myburgh 2015; Sadigh et al. 2014; Lan et al.) are two metrics that are commonly used to identify dangerous drivers. In typical social psychology studies, related items are introduced into user evaluation to ensure the robustness of the results. Therefore, Trajectory to Driving Behavior mapping evaluated four more driving behaviors apart from Aggressiveness and Carefulness, and those are listed in Table 2.

When an aggressive or careless driver is observed, depending on the position of that driver with respect to the targeted vehicle, the amount of attention that the driver of the targeted vehicle pays would still vary. Therefore, when evaluating the users' responses when driving as the targeted vehicle, the users are also asked to rate the four attention metrics listed in Table2.

Symbol	Description	Symbol	Level of Attention when
b_0	Aggressive	b_6	following the target
b_1	Reckless	b_7	preceding the target
b_2	Threatening	b_8	driving next to the target
b_3	Careful	b_9	far from the target
b_4	Cautious		
b_5	Timid		

Table 2: *Six Driving Behavior metrics (b_0, b_1, ...,b_5) and Four Attention metrics (b_6, b_7, b_8, b_9) used in user evaluation in obtaining the mapping*

Data-Driven Mapping

Trajectory to Driving Behavior Mapping conducts a user study that, has 100 participants identifying driver behaviors from videos. The trajectories of the videos are extracted from the Interstate 80 Freeway Dataset (Halkia and Colyar 2006). The users were asked to rate the metrics we listed in Table 2 on a 7-point scale and a 5-point scale for driving behavior and attention metrics, respectively.

After that, feature selection was applied to the results using least absolute shrinkage and selection operator (Lasso) analysis. In addition, the five features that are most appropriate for mapping to driving behaviors are extracted from ten potential ones. It concluded that using $\{s_{center}, v_{nei}, s_{front}, v_{avg}, j_l\}$ in mapping between features and driving behavior, and $\{s_{center}, v_{nei}, v_{avg}\}$ in the mapping between features and attention metrics can produce best regression models.

Using $\{s_{center}, v_{nei}, s_{front}, v_{avg}, j_l\}$ and $\{s_{center}, v_{nei}, v_{avg}\}$ as the features, linear regression is applied to obtain the mapping between these selected features and the drivers' behaviors. The results we obtained are below. For $B_{behavior} = [b_0, b_1, ..., b_5]^T$,

$$B_{behavior} = \begin{pmatrix} 1.63 & 4.04 & -0.46 & -0.82 & 0.88 & -2.58 \\ 1.58 & 3.08 & -0.45 & 0.02 & -0.10 & -1.67 \\ 1.35 & 4.08 & -0.58 & -0.43 & -0.28 & -1.99 \\ -1.51 & -3.17 & 1.06 & 0.51 & -0.51 & 1.39 \\ -2.47 & -2.60 & 1.43 & 0.98 & -0.82 & 1.27 \\ -3.59 & -2.19 & 1.75 & 1.73 & -0.30 & 0.61 \end{pmatrix} \begin{pmatrix} s_{center} \\ v_{nei} \\ s_{front} \\ v_{avg} \\ j_l \\ 1 \end{pmatrix} \tag{4}$$

Moreover, for $B_{attention} = [b_6, b_7, b_8, b_9]^T$,

$$B_{attention} = \begin{pmatrix} B_{back} \\ B_{front} \\ B_{adj} \\ B_{far} \end{pmatrix} = \begin{pmatrix} 0.54 & 1.60 & 0.11 & -0.8 \\ -0.73 & 1.66 & 0.63 & -0.07 \\ -0.14 & 1.73 & 0.25 & 0.15 \\ 0.25 & 1.47 & 0.17 & -1.43 \end{pmatrix} \begin{pmatrix} s_{center} \\ v_{nei} \\ v_{avg} \\ 1 \end{pmatrix} \tag{5}$$

We refer the readers to read (Cheung et al.) for more technical details and analysis.

Navigation

In this section, we describe how we leverage the benefits of identifying driver behaviors and ensure safe navigation. TDBM (Cheung et al. 2018) extends an autonomous car navigation algorithm, AutonoVi (Best et al. 2017), and shows improvements in its performance by using our driver behavior identification algorithm and TDBM. AutonoVi is based on a data-driven vehicle dynamics model and optimization-based maneuver planning, which generates a set of favorable trajectories from among a set of possible candidates, and performs selection among this set of trajectories using optimization. It can handle dynamic lane-changes and different traffic conditions.

The approach used in AutonoVi is summarized below: The algorithm establishes a graph of roads from a GIS database and computes the shortest global route plan using A* algorithm. Taking into account traffic rules and real-time traffic, the plan is translated to a static guiding path, which consists of a set of C^1 continuous way-points. AutonoVi then samples the speed and steering angle in a favourable range of values to obtain a set of candidate trajectories. Using the Control Obstacles approach, AutonoVi eliminates the trajectories that would lead to a possible collision. With the set of collision-free trajectories, AutonoVi selects the best trajectory using an optimization approach. It selects trajectories that avoid: i) deviating from the global route; ii) unnecessary lane changes; ii) sharp turns, breaking, and acceleration, which lead to discomforting experiences for passengers; and iv) getting to close to other road entities (including vehicles, pedestrians, and cyclists).

Neighboring Vehicles

AutonoVi proposed a proximity cost function to differentiate entities by class to avoid getting too close to other objects. It considers all vehicles as the same and applies the same penalization factor, $F_{vehicle}$, to them. Similarly, it applies higher factors : F_{ped} and F_{cyc} to all pedestrians and all cyclists, respectively. The original proximity cost used in AutonoVi is:

$$c_{prox} = \sum_{n=1}^{N} F_{vehicle}\, e^{-d(n)} \tag{6}$$

This cost function has two issues: i) it cannot distinguish dangerous drivers to avoid driving too close to them, and ii) it diminishes too rapidly due to its use of an exponential function. Therefore, TDBM proposed a novel proximity cost that can solve these problems:

$$c'_{prox} = \sum_{n=1}^{N} c(n) \tag{7}$$

$$c(n) = \begin{cases} 0 & \text{if } d \in [d_{t2}, \inf), \\ S_{TDBM} B_{far} \frac{d_{t2} - d(n)}{d_{t2}} & \text{if } d \in (d_t, d_{t2}], \\ S_{TDBM} \big[\frac{(d_t - d(n))(B_r - B_{far})}{d_t} + B_{far} \big] & \text{if } d \in (0, d_t]. \end{cases} \tag{8}$$

where $d(n)$ is the distance between the car navigating with TDBM and the neighbor n; d_t is a threshold distance beyond which neighbors will be applied with the 'far away' metric B_{far}; and d_{t2} is a threshold distance beyond which neighbors would not have any impact on TDBM's navigation. B_{far} and B_r refers to the attention metrics in Equation 5.

This proximity cost used in TDBM discouraged the optimizer from picking any candidate whose path is close to these dangerous drivers. However, this approach has a drawback: when the ego-vehicle and the neighboring vehicle are both slow, some unnecessary lane changing may occur. To avoid this, we add the relative velocity of the neighboring vehicle in relation to the ego-vehicle into the cost function. The new cost function also nullifies the effect of the cost on vehicles that are driving away from the ego-vehicle. The new cost function for vehicles is:

$$c'_{vehicle} = \sum_{n=1}^{N} max(0, v_{ego} - v_n) c(n) \tag{9}$$

where v_{ego} and v_n are the current progression speed along the lane of the ego-vehicle and the neighbor n respectively.

Pedestrians and Cyclists

The proximity costs for pedestrians and cyclists in AutonoVi and TDBM are still diminishing rapidly and do not take into consideration the velocity of the pedestrian/cyclist. We propose accounting for the current velocity in order to better predict and represent the zones to be avoided by the navigation algorithm:

$$c'_{obs} = \sum_{n=1}^{N} \frac{F(n)\, max(0, v_n \cdot \frac{\vec{s}_{ego} - \vec{s}_n}{||\vec{s}_{ego} - \vec{s}_n||})}{F(n) + ||\vec{s}_{ego} - \vec{s}_n||} \tag{10}$$

where $F(n)$ returns F_{ped} or F_{cyc} depending on the type of obstacle n. v_n represents the current normalized velocity of the pedestrian/cyclist. $\vec{s}_{ego}$ and $\vec{s}_n$ are the position of the ego-vehicle and the obstacle n, respectively.

Using these new cost functions, we can avoid drivers that are potentially riskier, stay away from pedestrians and cyclists more appropriately, and select a better navigation path.

Conclusion and future works

We present a new navigation approach leveraging the estimation of neighboring human drivers' behaviors and react to them accordingly. Using our approach, the navigation algorithm can more accurately estimate the level of awareness the ego-vehicle should have about neighboring vehicles, pedestrians and cyclists, and more effectively avoid those that require a higher level of awareness. Our approach can provide safer navigation among aggressive drivers, pedestrians, and cyclist and more efficient navigation when facing careful drivers.

The trajectory data that is currently available in the autonomous driving research community are limited, as labeling raw images are expensive. Currently, pedestrian and vehicle detection methods are advancing, and soon will be able to extract trajectory data reliably from raw data. The Trajectory to Driving Behavior Mapping applied in this work is based on highways, and the driving behaviors could be different in urban environment as there are pedestrians and cyclists involved. Furthermore, driving and pedestrians behaviors are different across countries and regions. With more data available, we would like to evaluate our approach on urban environments. Besides, there are works conducted to predict pedestrians trajectories (e.g., SocioSense (Bera et al.)), and we can combine them to navigate even safer around pedestrians and cyclists in the future.

References

[Aljaafreh, Alshabatat, and Al-Din] Aljaafreh, A.; Alshabatat, N.; and Al-Din, M. S. N. Driving style recognition using fuzzy logic. In *Vehicular Electronics and Safety (ICVES), 2012 IEEE International Conference on*, 460–463.

[Beck, Ali, and Daughters 2014] Beck, K. H.; Ali, B.; and Daughters, S. B. 2014. Distress tolerance as a predictor of risky and aggressive driving. *Traffic injury prevention* 15(4):349–354.

[Bera et al.] Bera, A.; Randhavane, T.; Prinja, R.; and Manocha, D. Sociosense: Robot navigation amongst pedestrians with social and psychological constraints. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, 7018–7025.

[Bera, Randhavane, and Manocha 2017] Bera, A.; Randhavane, T.; and Manocha, D. 2017. Aggressive, tense, or shy? identifying personality traits from crowd videos. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17.*

[Bergasa et al. 2014] Bergasa, L. M.; Almería, D.; Almazán, J.; Yebes, J. J.; and Arroyo, R. 2014. Drivesafe: An app for alerting inattentive drivers and scoring driving behaviors. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, 240–245. IEEE.

[Best et al. 2017] Best, A.; Narang, S.; Pasqualin, L.; Barber, D.; and Manocha, D. 2017. Autonovi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints. *arXiv:1703.08561.*

[Bojarski et al. 2016] Bojarski, M.; Del Testa, D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L. D.; Monfort, M.; Muller, U.; Zhang, J.; et al. 2016. End to end learning for self-driving cars. *arXiv:1604.07316.*

[Brill et al. 2009] Brill, J. C.; Mouloua, M.; Shirkey, E.; and Alberti, P. 2009. Predictive validity of the aggressive driver behavior questionnaire (adbq) in a simulated environment. In *Proceedings of* *the Human Factors and Ergonomics Society Annual Meeting*, volume 53, 1334–1337. SAGE Publications Sage CA: Los Angeles, CA.

[Buehler, Iagnemma, and Singh 2009] Buehler, M.; Iagnemma, K.; and Singh, S. 2009. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer.

[Cheung et al.] Cheung, E.; Bera, A.; Kubin, E.; Gray, K.; and Manocha, D. Identifying driver behaviors using trajectory features for vehicle navigation. In *Intelligent Robots and Systems (IROS), 2018 IEEE/RSJ International Conference on*.

[Cheung et al. 2018] Cheung, E.; Bera, A.; Kubin, E.; Gray, K.; and Manocha, D. 2018. Identifying Driver Behaviors using Trajectory Features for Vehicle Navigation. *ArXiv e-prints*.

[Dorsa Sadigh, Sastry, and Seshia 2017] Dorsa Sadigh, A. D. D.; Sastry, S.; and Seshia, S. A. 2017. Active preference-based learning of reward functions. In *Robotics: Science and Systems (RSS)*.

[Feng et al. 2012] Feng, Z.-X.; Liu, J.; Li, Y.-Y.; and Zhang, W.-H. 2012. Selected model and sensitivity analysis of aggressive driving behavior. *Zhongguo Gonglu Xuebao(China Journal of Highway and Transport)* 25(2):106–112.

[Geiger, Lenz, and Urtasun 2012] Geiger, A.; Lenz, P.; and Urtasun, R. 2012. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Goldman 2011] Goldman, D. A. 2011. Using a smartphone while driving can save your life. *Israel Mobile Summit*.

[Guy et al. 2011] Guy, S. J.; Kim, S.; Lin, M. C.; and Manocha, D. 2011. Simulating heterogeneous crowd behaviors using personality trait theory. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, 43–52.

[Halkia and Colyar 2006] Halkia, J., and Colyar, J. 2006. Interstate 80 freeway dataset. *Federal Highway Administration, U.S. Department of Transportation*.

[Harris et al. 2014] Harris, P. B.; Houston, J. M.; Vazquez, J. A.; Smither, J. A.; Harms, A.; Dahlke, J. A.; and Sachau, D. A. 2014. The prosocial and aggressive driving inventory (padi): A self-report measure of safe and unsafe driving behaviors. *Accident Analysis & Prevention* 72(Supplement C):1 – 8.

[Hoffmann et al. 2007] Hoffmann, G. M.; Tomlin, C. J.; Montemerlo, M.; and Thrun, S. 2007. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In *American Control Conference ACC'07*, 2296–2301. IEEE.

[Huang et al. 2017] Huang, S. H.; Held, D.; Abbeel, P.; and Dragan, A. D. 2017. Enabling robots to communicate their objectives. *arXiv:1702.03465*.

[Katrakazas et al. 2015] Katrakazas, C.; Quddus, M.; Chen, W.-H.; and Deka, L. 2015. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies* 60:416–442.

[Kolski et al. 2006] Kolski, S.; Ferguson, D.; Bellino, M.; and Siegwart, R. 2006. Autonomous driving in structured and unstructured environments. In *Intelligent Vehicles Symposium*, 558–563. IEEE.

[Krahé and Fenske 2002] Krahé, B., and Fenske, I. 2002. Predicting aggressive driving behavior: The role of macho personality, age, and power of car. *Aggressive Behavior* 28(1):21–29.

[Lan et al.] Lan, M.; Rofouei, M.; Soatto, S.; and Sarrafzadeh, M. Smartldws: A robust and scalable lane departure warning system for the smartphones. In *Intelligent Transportation Systems, 2009. ITSC'09. 12th International IEEE Conference on*, 1–6.

[Meiring and Myburgh 2015] Meiring, G. A. M., and Myburgh, H. C. 2015. A review of intelligent driving style analysis systems and related artificial intelligence algorithms. *Sensors* 15(12):30653–30682.

[Mohamad, Ali, and Ismail] Mohamad, I.; Ali, M. A. M.; and Ismail, M. Abnormal driving detection using real time global positioning system data. In *Space Science and Communication (IconSpace), 2011 IEEE International Conference on*, 1–6.

[Murphey, Milton, and Kiliaris] Murphey, Y. L.; Milton, R.; and Kiliaris, L. Driver's style classification using jerk analysis. In *Computational Intelligence in Vehicles and Vehicular Systems, 2009. CIVVS'09. IEEE Workshop on*, 23–28.

[Qi et al. 2015] Qi, G.; Du, Y.; Wu, J.; and Xu, M. 2015. Leveraging longitudinal driving behaviour data with data mining techniques for driving style analysis. *IET intelligent transport systems* 9(8):792–801.

[Sadigh et al.] Sadigh, D.; Sastry, S. S.; Seshia, S. A.; and Dragan, A. Information gathering actions over human internal state. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, 66–73.

[Sadigh et al. 2014] Sadigh, D.; Driggs-Campbell, K.; Puggelli, A.; Li, W.; Shia, V.; Bajcsy, R.; Sangiovanni-Vincentelli, A. L.; Sastry, S. S.; and Seshia, S. A. 2014. Data-driven probabilistic modeling and verification of human driver behavior.

[Sadigh et al. 2016] Sadigh, D.; Sastry, S.; Seshia, S. A.; and Dragan, A. D. 2016. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems*.

[Saifuzzaman and Zheng 2014] Saifuzzaman, M., and Zheng, Z. 2014. Incorporating human-factors in car-following models: a review of recent developments and research needs. *Transportation research part C: emerging technologies* 48:379–403.

[Shi et al. 2015] Shi, B.; Xu, L.; Hu, J.; Tang, Y.; Jiang, H.; Meng, W.; and Liu, H. 2015. Evaluating driving styles by normalizing driving behavior based on personalized driver modeling. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45(12):1502–1508.

[Treiber, Hennecke, and Helbing 2000] Treiber, M.; Hennecke, A.; and Helbing, D. 2000. Congested traffic states in empirical observations and microscopic simulations. *Physical review E* 62(2):1805.

[Ziegler et al. 2014] Ziegler, J.; Bender, P.; Schreiber, M.; Lategahn, H.; Strauss, T.; Stiller, C.; Dang, T.; Franke, U.; Appenrodt, N.; Keller, C. G.; et al. 2014. Making bertha drivean autonomous journey on a historic route. *IEEE Intelligent Transportation Systems Magazine* 6(2):8–20.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Predictions, Surprise, and Predictions of Surprise in General Value Function Architectures

Johannes Günther [1], Alex Kearney[1], Michael R. Dawson[1], Craig Sherstan[1] and Patrick M. Pilarski[1, 2]
[1]Departments of Computing Science and Medicine, University of Alberta, Edmonton, Alberta, Canada; [2]DeepMind
{gunther, pilarski}@ualberta.ca

Abstract

Effective life-long deployment of an autonomous agent in a complex environment demands that the agent has some model of itself and its environment. Such models are inherently predictive, allowing an agent to predict the consequences of its actions. In this paper, we demonstrate the use of General Value Functions (GVFs) for learning and representing such a predictive model on a robotic arm. Our model is composed of three types of signals: (1) predictions of sensorimotor signals, (2) measures of surprise using Unexpected Demon Error (UDE) and (3) predictions of surprise. In a proof-of-principle experiment, where the robot arm is manually perturbed in a recurring pattern, we show that each perturbation is detected as a jump in the surprise signal. We demonstrate that the recurrence of these perturbations not only can be learned, but can be anticipated. We propose that introspective signals like surprise and predictions of surprise might serve as a rich substrate for more abstract predictive models, improving an agent's ability to continually and independently learn about itself and its environment to fulfill its goals.

Introduction

Autonomous agents facing long-term deployment may encounter many challenges when interacting with the real world. The conditions of the environment and the agent itself may change over time. Further, it is impossible for engineers to fully anticipate all that such an agent must know ahead of time. The only way to overcome these shortcomings autonomously is for an agent to independently and continuously learn about itself and the environment in terms of its ongoing sensorimotor experience. One potential way to learn and represent information is to use predictions and predictive knowledge (Clark 2013). To this end, predictive models, such as General Value Functions (GVFs) (Sutton et al. 2011), present a method by which an agent might construct and represent information from its own experience. Such models should enable the agent to predict upcoming events and the outcomes of its actions, key information for successfully acting on its own. The usefulness of machine-made predictions has recently proven to be beneficial for various complex problems, even in challenging and changing environments. Examples include, but are not limited to, industrial laser welding (Günther et al. 2016), artificial limbs (Pilarski et al. 2013; Sherstan, Modayil, and Pilarski 2015) and robot navigation (Kahn et al. 2017). However, most research has focused on the use and prediction of signals generated by the environment (i.e., signals originating outside the agent, from the world or its physical body) and not *internal signals* (here defined as signals relating to the computational workings of the learning machine itself).

While knowledge about the environment is valuable for an autonomous agent to successfully interact with the environment on its own, further insight might be required to evaluate the consequences of the agent's actions. As stated by Schultz and Dickinson (2000, p. 476), "In general terms, learning can be viewed as the acquisition of predictions of outcomes (reward, punishment, behavioral reactions, *external stimuli, internal states*)" [emphasis added]. It is therefore necessary to not only learn about external sources of information but also about internal ones. Many authors have looked at using various internally generated metrics to drive exploration (White and White 2010; Gehring and Precup 2013), adapt algorithm parameters (White and White 2016; 2010; Sakaguchi and Takano 2004), adapt to changes in the reward function (White and White 2010), and minimize risk (Tamar, Castro, and Mannor 2016). Further, Sherstan et al. (2016) argued that internally generated signals, such as learning errors and statistical measures, should be made available to the agent as state information, enabling an agent to learn to make better decisions on its own. Learning external *and* internal signals by employing GVFs will result in a large number of predictions. Recent work has demonstrated the ability to learn a large number of online predictions for the sensor values of a mobile robot (Modayil, White, and Sutton 2014); in Pilarski and Sherstan (2016), a precursor to the present work, $\sim$18k GVFs were deployed in real time on the data stream of a robotic prosthesis.

In this paper we build on this prior work to provide an example of how GVFs can be used to make thousands of predictions about both external and internal signals at different time scales on a real-world problem domain. Using a proof-of-principle experiment, we learn thousands of predictions about incoming sensor readings provided by the sensors of a robotic artificial limb. Furthermore, we investigate measures that are related to these predictions to gain knowledge about the internal state of the prosthesis. One particular measure that we investigate in detail is the Unexpected Demon

Error (UDE) (White 2015). The UDE provides information about the comparison of the current prediction error to an average of previous errors. It can be seen as a measure of surprise, as it takes previous experiences into account and will only increase when current experience significantly differs from previous experience. Such a differing experience might be due to changing conditions, either in the environment or in the agent itself, providing important knowledge about the agent's functioning within said environment. We furthermore learn predictions about the UDE to provide the agent with a sense of how much surprise it might experience.

As a main contribution of the present work, we propose that predictions of raw perceptual data from an agent's data stream, along with sensations and predictions of surprise with respect to this data stream, can be used as a platform on which to build more powerful and more abstract predictive models of an agent's operation and interactions with its world. In the remainder of this paper, we demonstrate that such introspective information can be learned in a tractable, scalable way for use during long-term operation.

General Value Functions

As suggested, General Value Functions (GVFs) are a means to learn predictive knowledge (Sutton et al. 2011). A GVF v is defined in terms of the return, G_t. The return at time t is defined as $G_t = \sum_{k=0}^{\infty} \gamma^k C_{t+k+1}$, where C is the cumulant and γ is the discount rate. The cumulant is the signal of interest. The discount rate describes how future cumulants are weighted in the return. In the simplest case, $\gamma = 0$, the return is equal to the next cumulant. This setting is called myopic. As γ increases and approaches 1, future cumulants contribute more to the return. For $\gamma = 1$, the return is undiscounted and all future cumulants contribute equally.

A GVF v is defined as $v(s; \pi, \gamma, C) = \mathbb{E}_\pi[G_t | s_t = s]$. It maps from a state s to the expected return, given the agent follows the policy π and starts in the state s. The policy π specifies the behavior by providing the probability of taking an action a for a given state s. Together, the three parameters π, γ and C define what a GVF is about and are called *question parameters* (White 2015).

A way to learn General Value Functions is temporal-difference (TD) learning (Sutton 1988). TD learning allows for online and incremental computation of the value function by using estimates to make updates. This property makes it ideal to compute a sufficiently big number of GVFs to represent all information of interest. In this work, the value function is approximated by the inner product of a binary feature vector $x(s)$ that represents the sensor readings and a learned weight vector w. The value for a state is therefore computed as $v(s) = w^\top x(s)$. To update the value function, the TD error δ is computed after each time step as stated in line 3 in Algorithm 1. The TD error is then used to update the weights by taking a step towards the new estimate, based on the step size $0 < \alpha$. To potentially speed up learning by assigning credit to previously visited states, eligibility traces z are used. These traces decay according to the decay rate $\lambda \in [0, 1]$. The whole algorithm can be found in Algorithm 1 and an extensive introduction to TD learning can be found in Sutton and Barto (2018). The parameters α and λ are called *answer parameters*, as they define how the GVFs are learned. A collection of GVFs is called a *Horde* (Sutton et al. 2011).

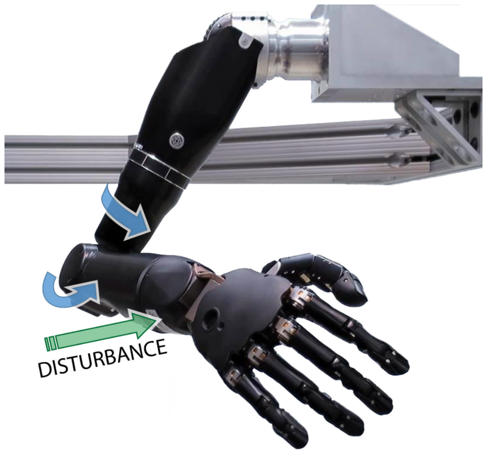


Figure 1: The Modular Prosthetic Limb (MPL) used for the experiments. The arrows indicate the nature of the repeated disturbance imposed during the experiment. The green arrow indicates the direction of the provided perturbation, while the blue arrows indicate the resulting joint movement.

Algorithm 1 TD(λ)

```
1: Initialize vectors $z \in 0^n$ and $w \in 0^n$; initialize a small
   scalar $\alpha$; observe state $s$
2: Repeat for each observation $s'$ and cumulant $C$:
3:     $\delta \leftarrow C + \gamma w^\top x(s') - w^\top x(s)$
4:     For $i = 1, 2, \cdots, n$:
5:         $z_i \leftarrow z_i \gamma \lambda + x_i(s)$
6:         $w_i \leftarrow w_i + \alpha_i \delta z_i$
7:         $s \leftarrow s'$
```

Unexpected Demon Error

One of the error measures we are most interested in for this paper is the Unexpected Demon Error (UDE) (White 2015). It provides a measure for unexpected changes in a signal due to changes in the environment. Mathematically, the UDE is calculated as

$$\text{UDE} = \left| \frac{\bar{\delta}^\beta}{\sqrt{\text{var}(\delta)} + \epsilon} \right|, \tag{1}$$

where $\bar{\cdot}^\beta$ is a moving average over the TD error δ and ϵ is a small constant to prevent division by zero. During learning, small changes in the TD error are to be expected, as the learner updates the value function and acquires knowledge about the world. The way the UDE is defined, it will neither react to the regular occurring learning nor to random noise, as both are considered in the mean and the variance of the TD error. The UDE will only significantly increase if the TD error behaves significantly differently due to changes in the

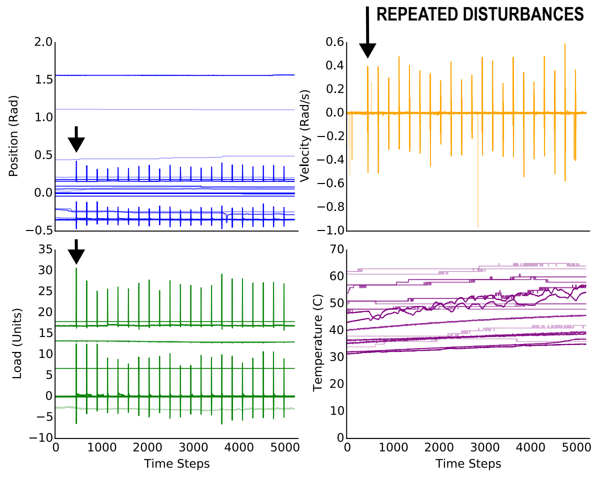


Figure 2: Decoded percept data from the robot over the 20min duration of the experiment. The 21 disturbances are clearly identifiable in data from the position, velocity and the load sensors. The temperature sensors show an increasing temperature over the experiment, with additional increases for some sensors due to the perturbations.

environment, triggering an unexpected amount of error, as the name implies. UDE can therefore very well be seen as a measure of genuine surprise and might provide important insight into the agent's learning.

Implementation and Experiments

Experimental Setup

The robotic arm used in the experiment is the Modular Prosthetic Limb (MPL v3) (Bridges, Para, and Mashner 2011), which can be seen in Figure 1. The MPL includes 26 articulated joints in the shoulder, elbow, wrist, and hand. Complex coordinated movements are possible via custom motors that provide up to 17 active degrees of freedom. Each of these motors is outfitted with sensors for load, position, temperature, and current. Additionally, each fingertip of the hand is instrumented with a 3-axis accelerometer and 14 pad pressure sensor arrays to provide a detailed sensor stream that is transmitted over a Controller Area Network bus and then sent to the control computer as User Datagram Protocol (UDP) broadcast packets. As one of the most advanced research prostheses currently available, the MPL provides a unique sensory-rich platform for testing GVF architectures.

When using sensor data for machine learning, the first decision to make is the choice of how infromation is represented to the learning machine. As in related work in the Atari Learning Environment (ALE) (Bellemare et al. 2013), we follow an approach of presenting the learning machine with raw binary data, prior to this data being decoded into real-valued numbers, states, or observations. Similar to the random access memory data used for learning on the ALE by Bellemare et al. (2013) and subsequent follow-up studies, the UDP packets that convey the MPL's data stream provide

Bit/Line	Data / sensor readings
0-39	Header
40-903	Position
904-1767	Velocity
1768-2631	Load
2632-3495	Temperature
3496-3520	Footer & Checksum

Table 1: UDP packet structure: the position of data in the MPL binary percepts (each bit corresponding to the respective line in the figures that follow).

a straight-forward, interpretable testbed with different dynamics for a system to learn about, including bits that do and do not vary with respect to perturbations, pseudo-random bits, and non-stationary bits that drift or slowly change activity over time (summarized in Table 1). As such, these bits—and not the real-valued signals that they help transmit—are the primary focus of our explorations below.

The experiment was conducted for approximately 20 minutes, resulting in 5217 time steps in total. For each time step, predictions, UDE and predictions of the UDE were computed and calculated, which took 0.23s per step on average. For the duration of the experiment, the arm was controlled to be compliant but motionless, streaming data regarding its current sensor readings: the position, velocity, load and temperature of all actuators. The experiment began with the arm remaining motionless in its resting position. This enabled the predictors to reach an acceptable level of accuracy. After two minutes, the arm was manually perturbed by an experimenter as indicated by the arrows in Figure 1, resulting in a change in the sensor readings for position, velocity and load. The arm then was allowed to move automatically back to its original resting position according to its compliance settings. The arm was perturbed in a similar way every minute for the remainder of the experiment, resulting in a recurring yet noisy pattern of sensory information. In total, 21 disturbances were recorded. The stream of 108 decoded sensor readings from this pattern are shown in Figure 2.

Predictive Architecture and Algorithmic Implementation

To create all signals of interest (predictions, surprise and predictions of surprise), at least two different hordes are necessary—one for the predictions and the surprise (Horde 1) and one for the predictions of the surprise (Horde 2). A third Horde (Horde 3) predicts the future values of the 108 decoded sensor values, and was added simply as an example of how predictions of raw data can be used as a substrate for more complex predictive questions. These Hordes are structured into two layers, as shown in Figure 3. Each Horde has two inputs, the cumulant, denoted as C, and the state vector, denoted as X. The first predictive layer receives the binary sensor signals from the MPL as state information and cumulant. As the signal consists of 3520 signals, there are 3520 GVFs in this Horde, one for each possible cumulant. The outputs of the first Horde are myopic ($\gamma = 0$) predictions about the binary inputs, and the predictions' UDE.

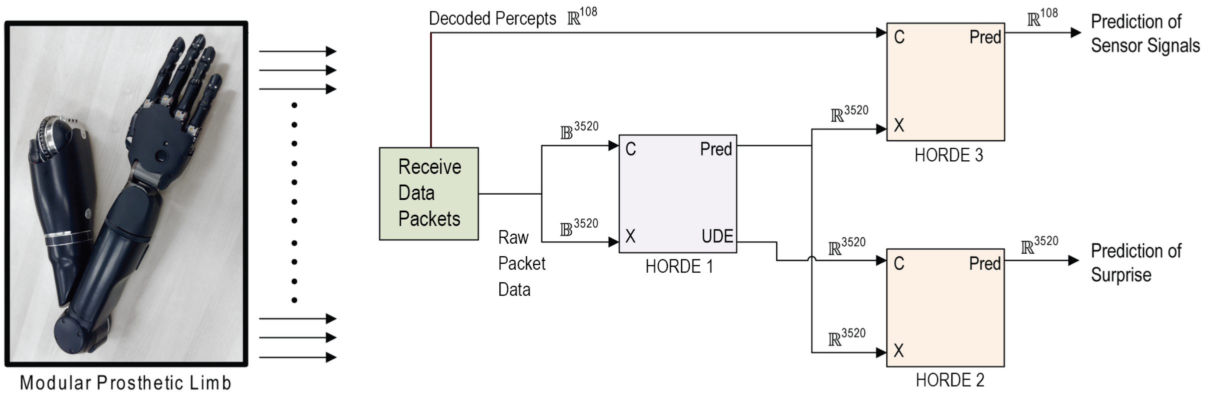


Figure 3: The prediction architecture used in the experiments. The sensor stream from the MPL on the left side is received over the network as a 3520 bit UDP packet, subsequently decoded into 108 floating point signals. These 3520 bits are delivered directly as both cumulant (C) and state (X) for Horde 1. The output of the first Horde is then fed into Horde 2 and 3 as the state X used in predicting the 108 decoded sensor signals and also to make predictions about the UDE (surprise) of Horde 1.

The outputs of the first predictive layer (Horde 1) are then used as inputs for the second predictive layer. There are two independent Horde architectures present in this layer (Horde 2 and 3). Horde 2 in the second layer receives the predictions from the first layer as state inputs $x(s)$ and the UDE as cumulants. As this layer predicts 3520 cumulants, it again consists of 3520 GVFs. Its outputs are predictions about the UDE of the first layer, with a discount rate of $\gamma = 0.999$, which corresponds to a prediction of 1000 steps into the future. The discount rate was chosen such that the predictions can reliably learn about the imposed perturbations, which are about 260 time steps apart.

Horde 3 receives predictions from the first layer as its state representation input $x(s_t)$, and its cumulants are the floating point decodings of sensor readings from the MPL. There are 108 GVFs in this Horde. Its outputs are predictions about the sensor signals, based on a discount or termination signal $\gamma = 0.9$, which corresponds to 10 time steps into the future.

For all predictive layers in this architecture, the same eligibility trace decay rate $\lambda = 0.99$ was chosen as a standard intermediate value of λ (Sutton and Barto 2018).

Experimental Results

To provide further intuition, we created synthetic data to demonstrate the expected behavior of the internal signals for a variety of potential external signal types, shown in Figure 4. Subplot (a) shows a potential data stream, including signals that do not change, recurring patterns, and random noise. Subplot (b) shows the predictions and should therefore match subplot (a), if the predictions are accurate. Subplot (c), which shows the UDE, should only spike for surprising changes in the original data and not for consistent noise. While the TD error for noise will constantly change, the UDE should only increase for the first occurrence of noise, as it keeps track of previous TD errors and will therefore expect TD errors of the same magnitude. It should furthermore not react to signals that are constant.

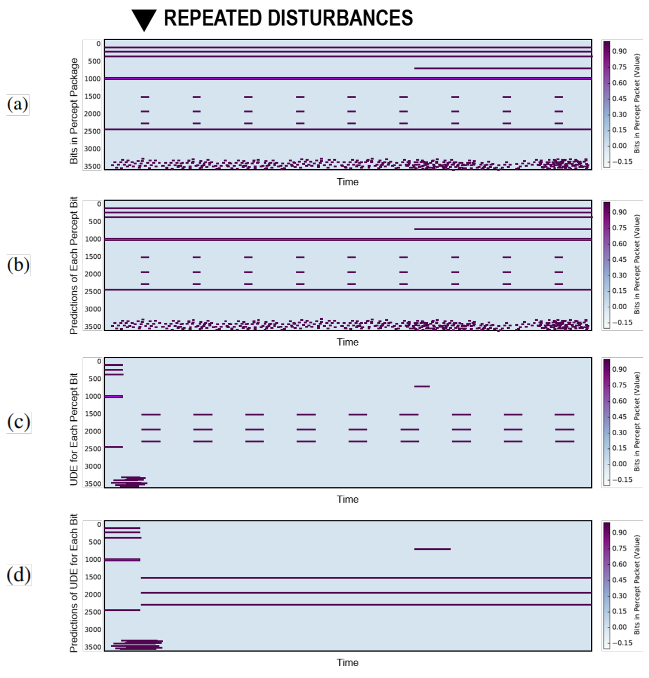


Figure 4: Simplified plots for the ideal relationship between (a) binary data, (b) predictions, (c) UDE and (d) predictions of UDE with $\gamma = 0.999$ for synthetic data.

The UDE should, however, react to recurring patterns, as the short moving average will forget about these signals over time. Subplot (d), which shows the predictions for the UDE, should show a longer activation where the UDE is active. The predictions of the UDE are only consistently active for the recurring pattern, as the predictions are consistently reinforced. The actual recorded data for all sensors over the whole duration of the experiment is shown in Figure 5.

Bits and Bit Predictions

To provide insight into the experimental results, Figure 5(a) shows the binary features of the data stream for all sensors. These features are created from the sensor readings by plotting the full contents of the UDP sensor packet received from the robot arm. Table 1 shows the line numbers of each sensor value in Figures 5, 6 and 7.

Purple bits are highly active, while light blue bits are not active, as indicated by the legend. Some bits do not change their value over time—this corresponds to constant sensor readings. For example, most sensor readings from the hand will be constant, as it is not moving during the experiment. Figure 6(a) shows a zoom in on the position and velocity bits for 200 time steps. As expected, most of the sensor stream is constant. However, around time step 4090 some of the values significantly change, as a result of the perturbation to the prosthetic arm. Other bits will be constantly changing. This may be due to sensor noise or due to inherently shifting signals, e.g. increasing temperature, or, in the case of the load sensors (Figure 7(a)), because the actuators need to keep the arm in place, resulting in the load sensors frequently being active and their values varying by small amounts.

As the predictions for the bits are myopic, they should ideally be the same as the actual bits. Figure 5(b) clearly shows that the predictions for the bits that show a constant behavior are identical. Even when zoomed in as shown in Figure 6(b) and 7(b), the bits that are constant are matching the predictions. The changing bits are not as trivial to predict. Bits that change randomly should in fact not be predictable and the predicted value should be distributed around the expectation, i.e. 0.5. Such random behavior can be seen in Figure 7(b) for some of the bits related to load between lines 2000 and 2050. For the position and velocity, shown in Figure 6(b) however, the predictions clearly map the disturbance, as can be seen by the changing predicted value, as the perturbation occurs around time step 4090.

UDE and UDE Predictions

To provide a meaningful measure of surprise, the UDE should show its highest activation both at the beginning of the experiment, when the sensor readings are new, and upon the disturbances, as the readings will significantly change when the arm is perturbed. Figure 5(c) clearly reveals the experimental design. The subplot shows the repetitive pattern of the arm displacement around lines 100, 950 and 1800. These binaries correspond to the position, velocity and load, respectively. Every time the arm is perturbed, the UDE significantly spikes as the sensor readings change. Noise still shows up in the UDE plot, but the intensity is lower, due to the UDE taking previous errors into account. The dampening of the noise is clearly displayed in Figure 6(d). Between lines 900 and 1000, some of the velocity binary features are highly volatile, as seen in subplot (a). The UDE, as shown in subplot (d), in comparison, only spikes twice, around time steps 4000 and 4090, where the actual displacements occur. Figure 7 elucidates a further aspect of the functionality of UDE: After the perturbation around time step 4310, the TD error in subplot (c) stays quite volatile until around time step 4460. The UDE decreases over this period and only spikes again when the TD error suddenly drops and stays low.

When looking at the predictions in Figure 5(d), it can be seen that the predictions about the UDE are not significantly active until the first disturbance occurs. After that, they are consistently high for the binaries that are affected by the perturbations. As the termination signal $\gamma = 0.999$ allows the predictions to consider 1000 time steps in expectation, the UDE predictions learn about the reoccurring movements and correctly predict the spikes in UDE. Figures 6(e) and 7(e) show in detail that the predictions anticipate that there will be changes in UDE due to perturbations, and at the same time filter the impact of UDE spikes that are not directly related, e.g. in lines 2000 to 2050 in Figure 7(e).

Discussion

This work presented the use of a predictive architecture to capture important information about the sensor stream of a prosthetic limb. The raw sensor stream of the MPL was received as binary values and served as an input to the first predictive layer that learned to predict these inputs in a myopic way and produced the Unexpected Demon Error (UDE) as a measure of the surprise with regard to the inputs.

In the original binary sensor data, the temporal structure of the perturbations is hidden by a significant amount of noise and general changes in the sensor values, for example due to changes in temperature. The (myopic) predictions of the sensor values match the original sensor values quite well for a large amount of the readings. However, some binary features behave randomly or almost randomly, resulting in predictions that are not accurate.

The UDE, however, is able to capture the perturbations and their effects on the position, velocity and load sensors. Each time the arm is manually moved, the surprise for each sensor peaks and falls afterwards. The UDE can therefore be seen as a valuable measure to inform the system about changes in its own functioning. Furthermore, the predictions about the UDE are consistently high after the first displacement, effectively capturing knowledge about the recurring pattern. At the same time, UDE and the predictions about the UDE are capable of filtering the noisy sensor readings to some degree, providing a better distinction between the perturbations and the normal, unperturbed running. For example, the UDE is consistently low for the checksum and the temperature, but spikes for signals that are impacted by perturbations of the arm. Intuitively, the system has learned about the potential changes in its functioning and to some degree can predict and expect these perturbations.

The internal signals that are generated by the suggested architecture can not only be thought of as direct inputs for a potential controller but can be looped in as additional context to improve the accuracy of these and other internal signals. For example, one could imagine using the predictions about surprise as additional context, incorporating unexpected motions into the agent's knowledge to improve its predictions of the sensor values. Including internal signals may improve the representation of the system, enabling the agent to learn more complex dependencies about itself and improve its performance autonomously.

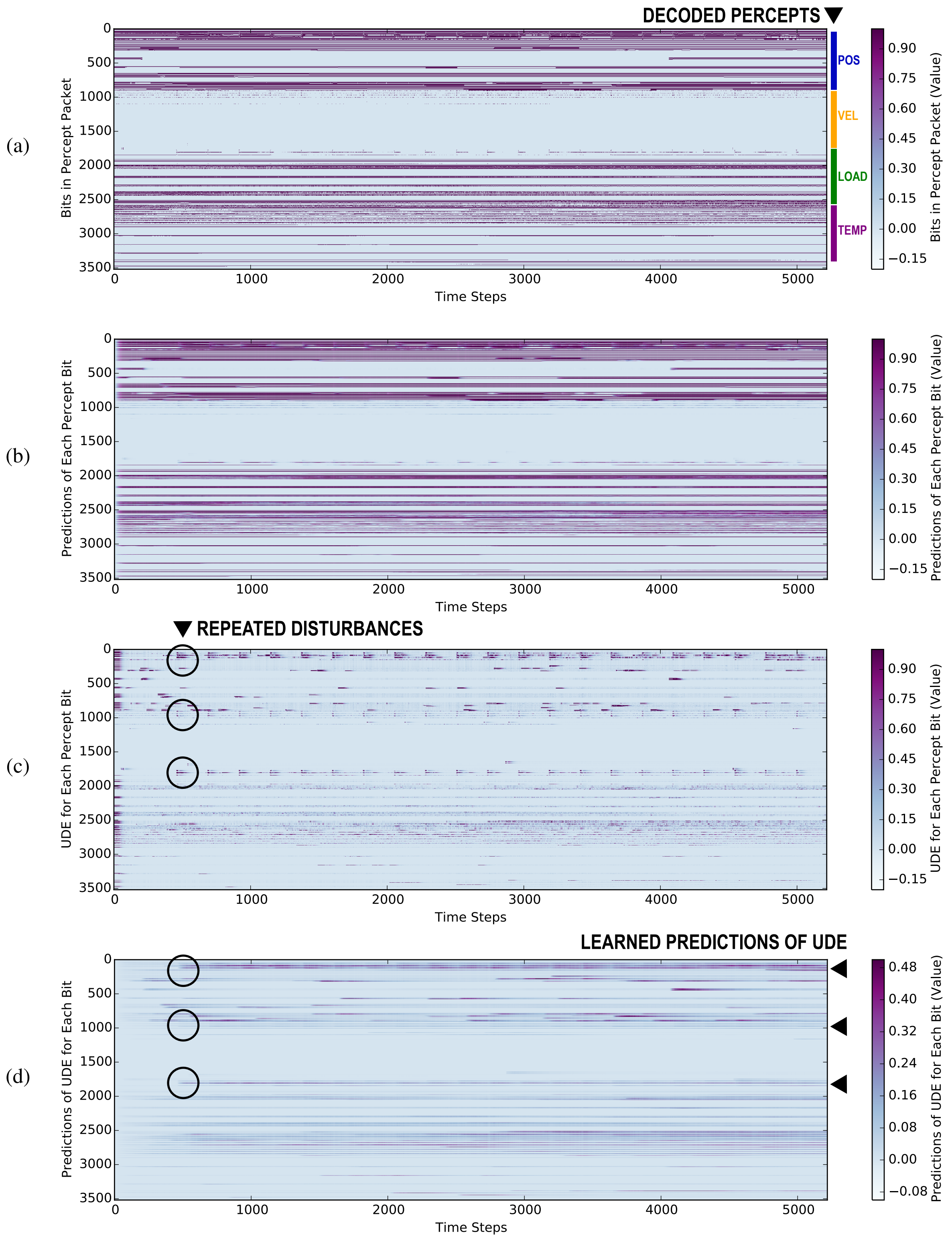


Figure 5: All recorded data for the experiment. The first subplot (a) shows the sensor stream from the MPL as decoded binaries. The second subplot (b) contains the myopic predictions for the binaries, provided by the first predictive layer. In the third subplot (c), the UDE is shown, followed by (d) the predictions about the UDE for a termination signal $\gamma = 0.999$.

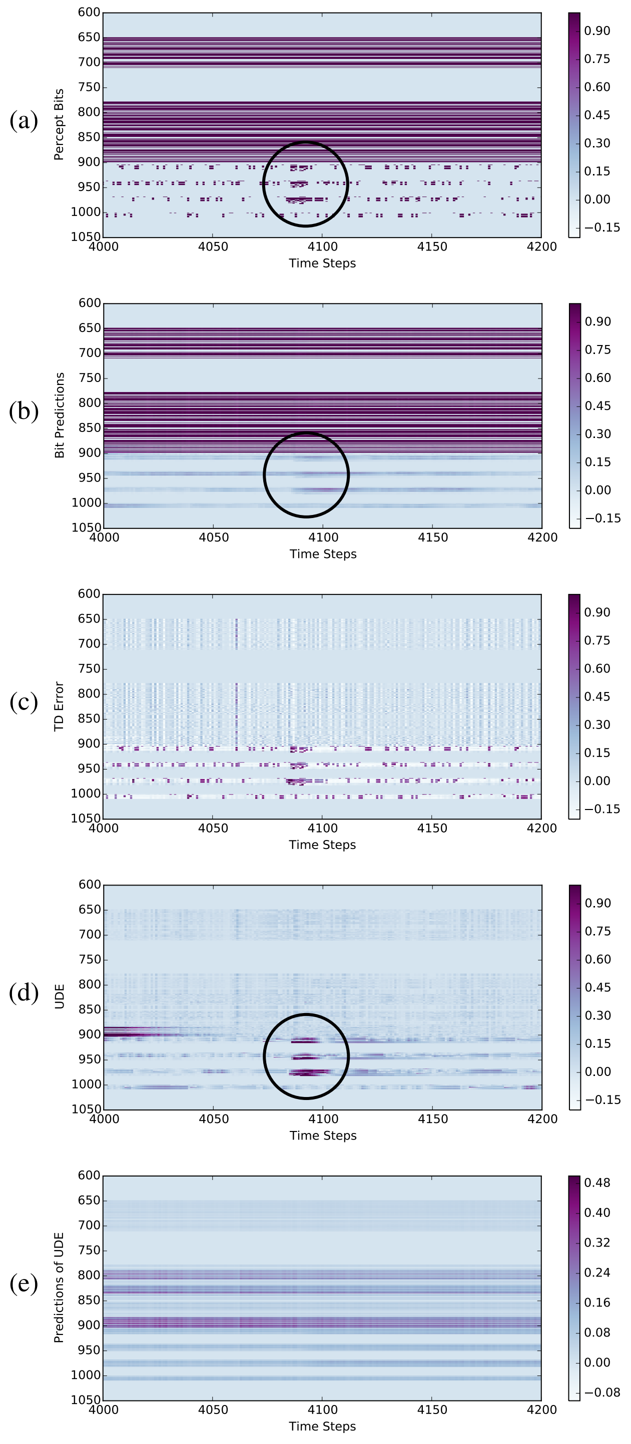


Figure 6: (a) Sensor data, (b) predictions ($\gamma = 0$), (c) prediction error, (d) UDE and (e) predictions of UDE ($\gamma = 0.999$) for position and velocity sensors.

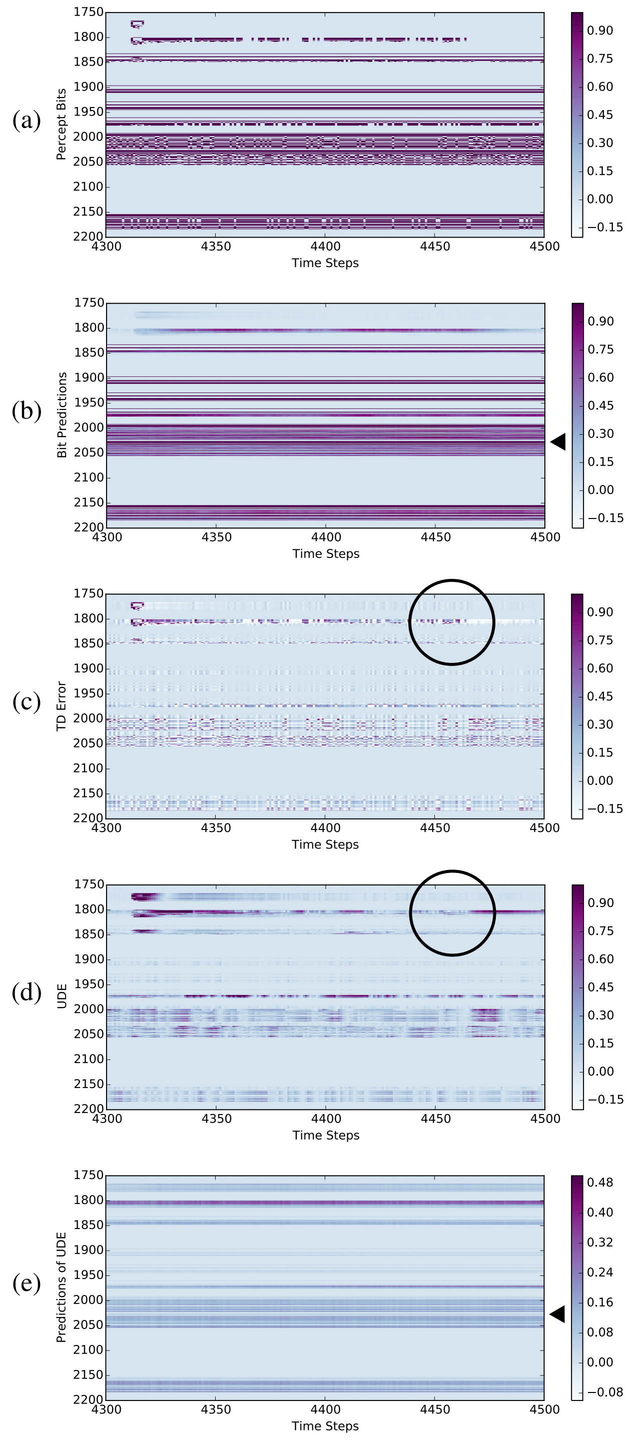


Figure 7: (a) Sensor data, (b) predictions ($\gamma = 0$), (c) prediction error, (d) UDE and (e) predictions of UDE ($\gamma = 0.999$) for load sensors.

Conclusion

The experiments in this paper were conducted to demonstrate how a predictive architecture can learn predictions, measure surprise, and learn predictions of surprise for a recurring pattern of sensor data from a prosthetic limb. The results show that important information about the underlying domain can be revealed by generating signals of interest from the ongoing operation of a Horde of General Value Function learners. The architecture in this paper learns surprise and predictions of surprise but does not make use of them. We suggest that the use of these signals in control learning is a natural extension that promises benefits: introspective signals can potentially help a learning agent to extend its knowledge not only about the environment but also about its own state within this environment. The present work can therefore be viewed as the process of learning a grounded, rudimentary model of actions and their consequences, which may create a foundation for learning more complicated concepts and relationships.

In the case of a learning artificial limb, predictions of surprise should provide knowledge of a change in the dynamics of the prosthesis before the change happens. If successfully learned, such predictions might serve as indicators not only of external variability like a new domain, a handshake, or unpredictable contact with objects, but also of changes in the function of the limb; the latter is a first step towards detecting the need for maintenance before the system breaks down. We suggest that introspective knowledge as presented in this work can be a valuable extension to systems that continually, autonomously learn and adapt in real-world settings.

Acknowledgements

This research was undertaken, in part, thanks to funding from the Canada Research Chairs program, the Canada Foundation for Innovation, the Alberta Machine Intelligence Institute, Alberta Innovates, and the Natural Sciences and Engineering Research Council. The authors also thank Kory Mathewson, Adam White, Richard Sutton, and Joseph Modayil for suggestions and helpful discussions, and Nadia Ady for her in-depth editorial feedback.

References

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

Bridges, M. M.; Para, M. P.; and Mashner, M. J. 2011. Control System Architecture for the Modular Prosthetic Limb. *Johns Hopkins APL Technical Digest* 30(3):217–222.

Clark, A. 2013. Whatever Next? Predictive Brains, Situated Agents, and the Future of Cognitive Science. *Behavioral and Brain Sciences* 36(3):181–204.

Gehring, C., and Precup, D. 2013. Smart Exploration in Reinforcement Learning Using Absolute Temporal Difference Errors. In *Autonomous Agents and Multiagent Systems (AAMAS)*, 1037–1044.

Günther, J.; Pilarski, P. M.; Helfrich, G.; Shen, H.; and Diepold, K. 2016. Intelligent Laser Welding Through Representation, Prediction, and Control Learning: An Architecture with Deep Neural Networks and Reinforcement Learning. *Mechatronics* 34:1–11.

Kahn, G.; Villaflor, A.; Ding, B.; Abbeel, P.; and Levine, S. 2017. Self-supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation. *arXiv preprint arXiv:1709.10489*.

Modayil, J.; White, A.; and Sutton, R. S. 2014. Multi-timescale Nexting in a Reinforcement Learning Robot. *Adaptive Behavior* 22(2):146–160.

Pilarski, P. M., and Sherstan, C. 2016. Steps Toward Knowledgeable Neuroprostheses. In *Proceedings of the International Conference on Biomedical Robotics and Biomechatronics*, 220–220. IEEE.

Pilarski, P. M.; Dawson, M. R.; Degris, T.; Carey, J. P.; Chan, K. M.; Hebert, J. S.; and Sutton, R. S. 2013. Adaptive Artificial Limbs: A Real-Time Approach to Prediction and Anticipation. *IEEE Robotics & Automation Mag.* 20(1):53–64.

Sakaguchi, Y., and Takano, M. 2004. Reliability of Internal Prediction/Estimation and Its Application. I. Adaptive Action Selection Reflecting Reliability of Value Function. *Neural Networks* 17(7):935–952.

Schultz, W., and Dickinson, A. 2000. Neuronal Coding of Prediction Errors. *Annual Rev. Neurosci.* 23(1):473–500.

Sherstan, C.; Machado, M. C.; White, A.; and Pilarski, P. M. 2016. Introspective Agents: Confidence Measures for General Value Functions. In *International Conference on Artificial General Intelligence*, 258–261.

Sherstan, C.; Modayil, J.; and Pilarski, P. M. 2015. A Collaborative Approach to the Simultaneous Multi-joint Control of a Prosthetic Arm. In *Proceedings of the International Conference on Rehabilitation Robotics*, 13–18. IEEE.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 2nd edition.

Sutton, R. S.; Modayil, J.; Delp, M.; Degris, T.; Pilarski, P. M.; White, A.; and Precup, D. 2011. Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 761–768. AAMAS.

Sutton, R. S. 1988. Learning to Predict by the Methods of Temporal Differences. *Machine Learning* 3(1):9–44.

Tamar, A.; Castro, D. D.; and Mannor, S. 2016. Learning the Variance of the Reward-To-Go. *Journal of Machine Learning Research* 17(13):1–36.

White, M., and White, A. 2010. Interval Estimation for Reinforcement-Learning Algorithms in Continuous-State Domains. In *Advances in Neural Information Processing Systems*, 2433–2441.

White, M., and White, A. 2016. A Greedy Approach to Adapting the Trace Parameter for Temporal Difference Learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 557–565.

White, A. 2015. *Developing a Predictive Approach to Knowledge*. Ph.D. Dissertation, Dept. of Computer Science, University of Alberta.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

SocialAnnotator: Annotator Selection Using Activity and Social Context

H M Sajjad Hossain and **Nirmalya Roy**
Departmenf of Information Systems
University of Maryland Baltimore County

Abstract

Precise and eloquent label information is fundamental for interpreting the underlying data distributions distinctively and training of supervised and semi-supervised learning models adequately. But obtaining large amount of labeled data demands substantial manual effort. This obligation can be mitigated by acquiring labels of most informative data instances using *Active Learning*. However labels received from humans are not always reliable and poses the risk of introducing noisy class labels which will degrade the efficacy of a model instead of its improvement. In this paper, we address the problem of annotating sensor data instances of various Activities of Daily Living (ADLs) in smart home context. We exploit the interactions between the users and annotators in terms of relationships spanning across *spatial* and *temporal* space which accounts for an activity as well. We propose a novel annotator selection model *SocialAnnotator* which exploits the interactions between the users and annotators and rank the annotators based on their level of *correspondence*. We also introduce a novel approach to measure this *correspondence* distance using the *spatial* and *temporal* information of interactions, type of the relationships and activities. We validate our proposed *SocialAnnotator* framework in smart environments achieving $\approx 84\%$ statistical confidence in data annotation.

Introduction

Acquiring labeled data instances is an important task for training supervised and semi supervised machine learning models. In most of the problem domains, both domain knowledge and label information for a learning algorithm are compiled by the human annotators. As a result human intervention is indispensable for collecting ground truths. Labeling large amount of data suggests engaging more domain experts or extending the time for the labeling process. Adapting either of these approaches is a daunting task as it is difficult to find abundant domain experts who can relentlessly provide labels. Consider building an Activities of Daily Living (ADLs) classifier using accelerometry data. If the sampling frequency is 30 Hz and we collect data from a single user for a single day, we end up with approximately 2.5 million data instances. Moreover, the reliability, availability of domain experts, and the incurring costs associated with data annotation process makes it a painstaking step while building a machine learning model. It is possible to reduce the complexity of data annotation by dissecting the problem domain and identifying the relevancy of data with appropriate activity. For example, in case of ADLs we can select a handful number of activities or emphasize more on a specific period of the day instead of considering all of the data. From a machine learning perspective, we can view this as to look for most important data instances which can have significant impact on our classifier. By utilizing Active Learning (Bodó, Minier, and Csató), we can select the most informative data instances and pose the label queries to the annotators. There are alternative methods to reduce annotation effort other than Active Learning like utilizing *Crowdsourcing* platforms (Love and Hirschheim) or training the learning model using unlabeled data (Fiorini, Cavallo, and et al. 2017) (Gjoreski and Roggen 2017). However these approaches can invoke negative impact, for example, annotators in Crowdsourcing platforms are mostly not domain experts and can introduce noisy labels in the model.

Activity recognition using wearable and ambient sensors in smart home domain is a well studied problem in literature (Guan and Plötz 2017)(Shoaib et al. 2015). Existing activity recognition methods endure limitation in terms of data scarcity and scalability. The sensors produce an immense volume of data due to high sampling frequency in order to capture fine-grained information without any loss. In order to collect ground truth information, existing works have relied on the video feed heavily where each video frame is mapped with the timestamp (Hossain, Khan, and Roy 2017a). In this paper, we propose an online annotator selection model while exploiting active learning in smart home activity recognition domain. Even though active learning can be effective in acquiring labels, its foundation is built on impractical assumptions - *an annotator who is always available to provide the correct labels to every queries without incurring any cost and the active learner can query as many instances as possible* (Donmez and Carbonell 2008). In practical, a single annotator may or may not respond to all of the queries. Therefore, exploiting multiple annotators seems more practical (Yang and Wooldridge 2015), nevertheless their expertise level may differ drastically. Moreover, the labels received from these imperfect annotators are not always reliable, so if we pose an important query to a wrong annotator all the efforts will be pointless. Thus based on the informativeness of the selected data instance, it is always desirable to pose the query to the right annotator.

We first attempt to model the human relationships and assemble a knowledge base to represent the level of influence to others according to social and physical context. We formulate a corresponding distance metric using this knowledge base which expresses the level of *correspondence* between connected users given the current context. While calculating this distance, we also consider the distance between activity space for a certain user. Activity distance enables us to find out similar activities with respect to their spatial and temporal properties. For example, if a person eats and watches television in the living room most of the time, then *eating* and *watching television* has similar spatial property. In such cases if an annotator can label *eating* activity efficiently, then our assumption is that he can also effectively provide label for *watching television* activity. In this paper we design an annotator selection model *SocialAnnotator* based on Contextual Multi-Armed Bandit (CMAB) algorithm where the *context* resembles the features of the queried instance and action corresponds to the annotator selection. We consider the time, space, prior context history and the approximate label received from the learner as the *context* information for CMAB. SocialAnnotator works in a collaborative manner where the connected users collaborate and provide label for each other.

Related Work

Activity recognition has been one of the core research areas in ubiquitous computing field for many years (Lara and Labrador 2013) (He et al. 2008). This rapid surge and advancement in learning activity pattern have also assisted a plethora of application domains ranging from sports (Daiber and Kosmalla 2017) (Hossain, Khan, and Roy 2017b) to health analytics (Samarah et al. 2017). Activity recognition research have been addressed from two pespectives using *computer vision* (Li and Vasconcelos 2017) (Jalal et al. 2017) and *sensor modalities* (Hossain, Khan, and Roy 2017a) (Davila, Cretu, and Zaremba 2017). Various machine learning models including both shallow learning (Lee and Cho) (Wyatt, Philipose, and Choudhury 2005) and deep learning (Guan and Plötz 2017) (Bhattacharya and Lane 2016) algorithms have been exploited in existing activity recognition literature over the years. Activity recognition models exploiting supervised and semi-supervised learning algorithms have to heavily rely on the number of labeled data instances. Some literature have proposed models using unsupervised learning algorithms (Twomey et al. 2017) (Münzner et al. 2017) (Bouchard, Bouchard, and Bouzouane 2012) but if the distribution of the data is not clearly inherent, unsupervised algorithms fail to find the pattern in the data.

To address the problem of gathering ground truth information, active learning has been employed by few researchers. The authors of (Bagaveyev and Cook 2014) investigated several active learning approaches in smart home activity recognition context and evaluated with real world data set. Diethe et al. proposed a bayesian approach by utilizing active and transfer learning in (Diethe, Twomey, and Flach 2016). In (Liu, Chen, and Huang 2010) and (Stikic, Van Laerhoven, and Schiele 2008) the authors exploited uncertainty based active learning. (Ho et al. 2009) used an entropy based approach to measure the informativeness of data instances. In our previous work (Hossain, Khan, and Roy 2017a) we proposed a clustering based heuristic to find the most informative instances. Hasan et al. proposed a context aware model using active learning (Hasan and Roy-Chowdhury 2015). They utilized entropy and mutual information of the instances to filter out the most informative data instances. (Xu et al. 2014) applied active learning in a contextual multi-armed bandit setting to do the activity classification. However, while employing active learning it is not always guaranteed to receive correct and noise free labels (Donmez and Carbonell 2008). In this paper we take a radically different approach than the existing literature and focus on improving the impact of active learning by selecting proper annotator using social relationships.

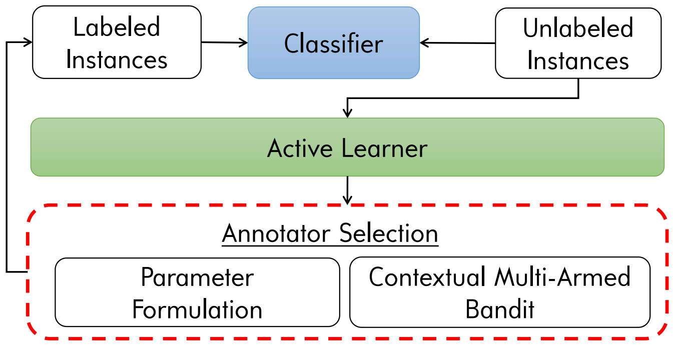


Figure 1: *A high level structure of the modules in SocialAnnotator framework.*

Overall Framework

SocialAnnotator framework is composed of three major components. We have an activity recognition classifier which is trained on labeled data instances from wearable devices that provide raw accelerometer data. After building a stable classifier, we start feeding unlabeled instances and predict the class label. We then start filtering uncertain data instances from the stream of unlabeled data instances in our *Active Learner* module. In our active learner module, we measure the entropy of the instances and select the instance with maximum entropy. We then send the selected instance to our *Annotator Selection* module. Note that in our daily life we interact with a number of people. The interaction can be physical or virtual through social network but every interaction is an opportunity to observe and share information. The key insight here is that we are connected and have more interactions with the people who we are related with us. These connected people might be direct witnesses of what we are doing in our day to day life. As a result these social relationships and correspondence lead us to have knowledge about the activity patterns of the people we are connected with.

We model the level of correspondence using a distance parameter. We calculate the *spatio-temporal* distance between two connected users using their probability distribution of location. The *spatio-temporal* distance lets us know about the intersection between their location distribution. We also incorporate a weight based on the strength of the relationship. The people with whom we interact more have higher potential to know about our daily routines. After formulating the distance parameter, we model a budget constrained context aware multi-armed bandit. The task of the bandit is to select annotator given the distance parameter and context. We design the bandit in such a way so that it does not act in a greedy way by introducing costs associated with each annotator and a budget constraint. We adapt a game theoretic approach where we have to ensure maximum gain and keep track of our budget as well. The costs associated with the annotators are not static, as the level of interaction evolves over time. For example, we have regular interactions with the people at our work place during the week days, but over the weekend we tend to mingle with close friends. Also a person can be connected through multiple relationships (close friend and colleague in a work place at the same time). Therefore, we consider the cumulative relationship weights of all the relations for quantifying the level of correspondence between two users. Figure 1 depicts a conceptual structure of our *SocialAnnotator* framework.

Distance Metrics

In this section we discuss the metrics which correlate our annotators with activities. We collect raw accelerometer and location data from the users and formulate the following distance metrics using the temporal and spatial information of these sensor modalities.

Spatio Temporal Distance

We calculate the spatio-temporal distance of the related users when an activity is performed. This distance metric implies if an user has any knowledge about the performed activity by another user he or she is related to. While computing this parameter, we also consider the neighboring locations of where the activity was performed. We calculate the likelihood of each related user j being in a location l_i given the current context x_t. We process this as a *categorical distribution*. Let us consider a set of activities W with whom the user u_i is connected. The location of each activity is an observation of our distribution, and the location set $L(W)$ is a sample of that distribution with cardinality m. Each location in $l_i \in L(P)$ has a prior probability. We denote the probabilities of locations as vector $p = (p_1, p_2, p_3 ... p_m)$. Let us consider q be the location probability distribution of an annotator a_i who is connected to user u_i through a relation. We then calculate the conditional distributions of p and q given context x_t and time t. Using these conditional distributions we calculate the distance between them using *Bhattacharyya distance* (Bhattacharyya). The distance between these two conditional distributions is defined as:

$$\begin{aligned} d_{st}(p(x,t), q(x,t)) &= -ln(\mathbb{B}(p(x), q(x))) \\ &= \sum_{i=1}^{m} \sqrt{p_i(x,t) q_i(x,t)} \end{aligned} \tag{1}$$

In Eqn 1, $\mathbb{B}$ is the *Bhattacharyya coefficient* which provides the measurement of overlap between the two probability distributions. This distance provides us information regarding the annotators who reside closer to the user. We calculate this spatio-temporal distance for all the connected users and take the annotators who were closest to the vicinity where the activity was performed. If no annotator was present in the vicinity where the user performed the activity, we assume that annotators dwelling in the neighboring locations may have knowledge about the activity label.

Activity-Activity Distance

We exploit the connectivity among activities to filter appropriate annotators. Our intuition is that if the properties of an activity W_i prevails in a similar spatial and temporal space to another activity W_j and an annotator a_k has efficiently provided reliable labels to activity W_j then a_k is a potential annotator who can provide label of activity W_i. To calculate this distance we consider three components of an activity pair - $correlation$, $spatial$ and $temporal$. Correlation calculates the co-occurrence frequency of the activity pair, the spatial and temporal component models the probability of an activity pertaining to the same location and time constraints. The distance is defined as:

$$\begin{aligned} d(w_i, w_j) = f(w_i, w_j) \mathcal{N}(||t_{a_i} - t_{a_j}||^2, \mu_t, \sigma_t) \\ \mathcal{N}(||l_{a_i} - l_{a_j}||^2, \mu_s, \sigma_s) \end{aligned} \tag{2}$$

In eqn 2, $f(w_i, w_j)$ denotes the co-occurrence frequency between a pair of activity, $l_{a_i}, l_{a_j}, t_{a_i}, t_{a_j}$ are the spatial and temporal parameters of the associated activities.

Relationship Weight

The strength of the social relationship can be integral in selecting annotator. There may not be any annotator who directly witnessed the user doing an activity. However, human being follows a cognitive routine most of the time and the persons mostly associated with his life are acquainted with the routine. For example, the family members living with the user are usually more familiar with his routine. Some annotators can also be remotely connected (e.g. updates on social network, talking over the phone or even playing online games together). So certain relationships provide more emphasis and demand more attention while choosing the annotator. For this reason, we try to provide weight to each connected user according to the relation. However this weight can not be static for all of the users as in real life not all relationships are same and they evolve over time. For example, consider the relationship with your office colleagues, initially they could be just colleagues but over time some might become your close friend. On the other hand one might be in touch with their parents on regular basis, but a different person might not. So for each person the weight of relationship is different. We use the relationship intensity strength proposed in (Srba and Bieliková 2010) to model our relationship weight. The interaction between two users (e.g. phone call, messaging, meeting etc.) or shared information (e.g. playing soccer together, common hobby) are designated as “rate factor”. Depending on the social aspect these rate factors regulates the strength of a relationship. The partial relationship weight between user k and j for one factor is defined as:

$$Y_f(k,j) = \frac{\omega_{kj} \sum_{i=1}^{l} f_t}{1 + ln(1 + l_c)} \tag{3}$$

In eqn 3 ω_{kj} is the weight of the rate factor, l is the count of rate factors, l_c is the count of instances of the rate factor and f_t models the time influence. The final weight $Y(k,j)$ is measured by taking the arithmetic mean of the partial weights of all the rate factors.

Now that we have formulated all our distance metrics, we now define our final user to user distance metric. The activity-activity distance metric provides the distance between activities and finds the similarity among them. We maintain the count of such activities for which the annotators have performance score more than a pre-defined threshold δ. We utilize this count as an additional weight W_c for the annotators. The final distance is calculated using the following equation:

$$D(k,j) = Y(k,j)\, W_c\, d_{kj}\Big(p(x,t), q(x,t)\Big) \tag{4}$$

Methodology

In this section, we discuss active learning, the contextual multi armed bandit problem and the modeling of arms or actions of the bandit, the rewards and the context of our problem domain.

Active Learning

Active learning is fitting for problems pertaining to large amount of unlabeled data instances. In the context of activity recognition using wearable devices, we have to process overwhelming number of data instances which makes active learning befitting. We only label the data instances which provide highest gain which is reducing the generalized error of our classifier. In our proposed model we propose to use Active learning using pool-based sampling as we receive a stream of data in a very short period of time. We select a data instance from a pool of instances in a greedy way. Queries are typically conforming to the measure of uncertainty. Here our assumption is that the instances which are least certain are close to the decision boundary and labeling these instances will provide maximum gain. To measure the uncertainty we calculate the entropy of the provided instances and query the instance with maximum entropy. We calculate the maximum entropy and select an instance by following equation

$$\begin{aligned} x_H &= \underset{x}{\operatorname{argmax}}\, H_\theta(Y|x) \\ &= \underset{x}{\operatorname{argmax}} - \sum_y P_\theta(y|x) log\, P_{theta}(y|x) \end{aligned} \tag{5}$$

Contextual Multi-Armed Bandit

A contextual bandit problem is composed of N arms or actions. In our context an action refers to selecting an annotator. Our goal is to maximize this reward in each iteration. However, by selecting a sub set of the actions in a regular manner might always provide maximum reward. For example, a person's spouse or close friend has better idea about his daily activity routine than any one else. So by selecting the spouse or close friend in each round will maximize the reward outcome. If we consider the annotators as resources, prompting the same set of annotators will lead to resource exhaustion. In order to tackle this, we introduce a resource constraint or budget for each of the annotator. The annotators who ensures higher potential reward, incur higher cost. As a result given an overall budget our aim is to maximize the total reward while ensuring aggregated resource consumption remains bounded by a given budget. Let us denote the action set as $\mathcal{A} = \{a_1, \cdots, a_k\}$. We consider the cardinality of $\mathcal{A}$ to be finite as an user is connected to a finite number of people. A d-dimensional feature vector $x_t \in \mathcal{X}$ denotes the context information received at time t. At each time t, an agent or policy π decides to choose an action a_i based on the context x_t and receives reward r_i^t. The history of taken actions and received reward is denoted by $\mathcal{H}_{t-1} \rightarrow \{a_i(\tau), r_\tau, x_i(\tau)\}$ for $i = 1, \ldots, N$ and $\tau = 1, \ldots.., t-1$ where $a_i(\tau)$ denotes the chosen action which generated reward r_τ. The reward of an action is generated from an unknown distribution regulated by the given context. Let us consider the optimum action at t is a_i^* and its corresponding reward is $\tilde{r}_i^t$.

We want to select the action which results in reward close to the optimum one, so the aim is to maximize the reward in each step and minimize the difference between the overall optimum reward and the reward received. The difference between the optimal reward and the aggregated reward received is called *regret*. We provide a formal definition of *regret* as following

$$\mathbb{R} = \sum_{t=1}^{T} \mathcal{R}_t = \sum_{t=1}^{T} (\tilde{r}_i^t - r_i^t) \tag{6}$$

In this eqn, $\tilde{r}_i^t$ is the optimum reward at step t and r_i^t is the reward received. Let us define our reward function as $r_t = f(x_t, a_i(t))$, where $f(x_t, a_i(t))$ is the reward mapping function for arm $a_i(t)$. In order to maximize the reward function, the agent needs to learn the underlying function f which maps the context to action. In order to acquire knowledge about the latent function f, the agent has to explore other actions instead of choosing the optimum action which provides the best outcome. ϵ is our exploration parameter. The predictive distribution of our reward function depends on the current context and the history of actions taken. This is a normal distribution with mean μ_r and variance V which are defined as following

$$p_\theta(r_t|\mathcal{H}_{T-1}, x_t) = \mathcal{N}(\mu_r(t), V_t) \tag{7}$$

Each action $a_i(t)$ is also associated with a cost $c_{a_i}^t$. The cost associated with an annotator is variable in each round as the distance between users defined in eqn 4 varies over time. The costs are independently and identically drawn from an unknown continuous distribution with mean μ_c. We adhere to the same settings in (Xia, Qin, and et al.): (i) the rewards of an action are independent of its costs (ii) the rewards and costs of an arm are not influenced by other actions (iii) the rewards and costs of an action are independent and identically distributed at each iteration. Let us define a known parameter, *budget* B which designates the number of time the algorithm can invoke annotators. This budget constrain also helps us to supervise the stopping time $t_s(B)$ of our algorithm which is defined as following

$$\sum_{i=1}^{t_s(B)} c_{a_i}^i \leq B < \sum_{i=1}^{t_s(B)+1} c_{a_i}^i \tag{8}$$

Let us denote $\mathcal{R}^*$ as our optimum aggregated reward at stopping time $t_s(B)$. We calculate the expected regret, evaluated over the randomness of rewards and costs by modifying eqn 6.

$$\mathbb{R} = \mathcal{R}^* - E\Big[\sum_{t=1}^{t_s(B)} (r_{a_t}^t)\Big] \tag{9}$$

Actions The action space for an user is proportional to the number of connected annotators. An action corresponds to selecting an annotator from the correspondence vector M. Each element $m_{ij} \geq 0$ is congruent to how relevant the annotator is with respect to the user in terms of our distance metric and labeling accuracy. The expected reward to cost ratio of an annotator a_i is $\rho_{a_i} = \frac{\mu_r^{a_i}}{\mu_c^{a_i}}$. According to (Xia, Qin, and et al.), if both reward and cost distribution of an action is known, pulling the arm with maximum ρ can provide the expected reward as the optimal algorithm. When the distributions are unknown, we should select the annotator with the maximum ρ and also ensure exploration on the other rarely selected annotators.

Context A context vector x_t portrays the features and characteristics of each annotator. The features considered in a context vector are the timestamp t, location s, n performance metrics of the annotator with respect to each activity $c_1, \ldots, c_n$. We do not include the sensor data in the context vector.

Reward Our reward mapping function randomly generates reward according to the conditional probability measure defined in eqn 7. Initially the model is uncertain about the value θ. Our reward mapping function f is defined to measure the reduction in variance of our classification model between two iterations. For making things simple, our objective is to minimize the squared loss of the true label and the label received from an annotator. We define our expected error as following

$$\mathbb{E}\Big[(\hat{y} - y)^2 | x_t, y_l\Big] = \mathbb{E}_{Y|x}\Big[(y - \mathbb{E}_{Y|x}[y|x, y_l])^2\Big] + (E_L[\hat{y}] - \mathbb{E}_{Y|x}[y|x])^2 + E_L\Big[(\hat{y} - E_L[\hat{y}])^2\Big] \tag{10}$$

In eqn 10 $\mathbb{E}_L[.]$ is the expectation over the labeled training set L, $\hat{y}$ is the label received from an annotator and y is the true label of the instance. $\mathbb{E}_{Y|x}\Big[(y - \mathbb{E}_{Y|x}[y|x])^2\Big]$ indicates noise or uncertainty of y given x. The second term represents bias which is the error due to the selected action. The third term represents the output variance of our model. Therefore minimizing the variance will ensure to minimize the generalization error of our model. So we try to reduce error by selecting annotators that establish highest variance reduction of our activity recognition model. For any action a_i, number of times it is invoked $n_{a_i,t}$, average cost $\bar{c}_{a_i,t}$ and average reward $\bar{r}_{a_i,t}$ and the exploration parameter is $\epsilon_{a_i,t} = \sqrt{\frac{2log(t-1)}{n_{i,t}}}$. We calculate index $D_{a_i,t}$ for each annotator:

$$J_{a_i,t} = \frac{\bar{r}_{a_i,t}}{\bar{c}_{a_i,t}} + \frac{\bar{r}_{\epsilon_i,t}}{\bar{c}_{a_i,t}} + \frac{\bar{r}_{\epsilon_i,t}}{\bar{c}_{a_i,t}} D(k,i) \tag{11}$$

In eqn 11, the average reward to cost ratio represents the exploitation. The first influences our algorithm to choose the arms with higher rewards. The exploration term $\frac{\bar{r}_{\epsilon_i,t}}{\bar{c}_{a_i,t}}$ favors the annotators who provide less reward and as a result invoked infrequently with lower costs. Exploring weaker annotators may be conducive as our budget is limited. The final term enforces joint exploitation and exploration. Our whole methodology is summarized in Algorithm 1.

Algorithm 1 *SocialAnnotator* Annotator Selection

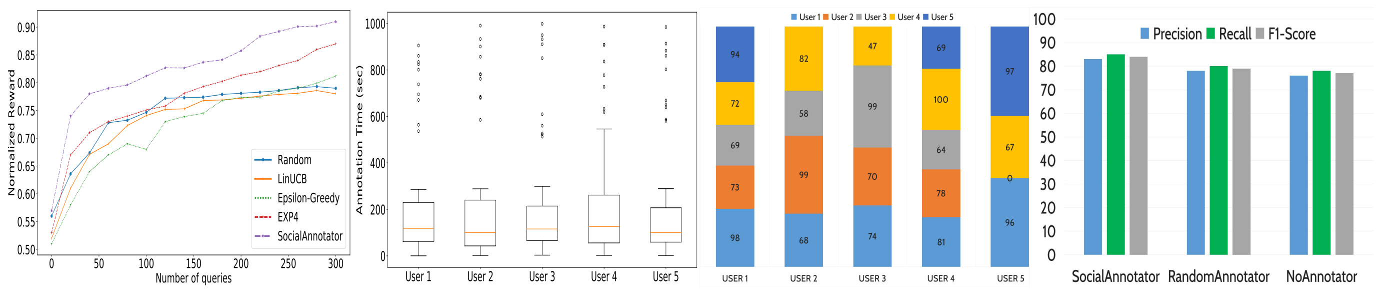
```
Require: $U$, A pool of unlabeled instances $\{(x)^u\}_{u=1}^U$,
    $A = \{a_1, a_2, \ldots a_k\}$, A list of connected annotators
 1: Output: Best annotator $a_i$.
 2: Select instance $x_t$ with maximum entropy
 3: $p \leftarrow$ location probability distribution of the user
 4: Reward Index $J \leftarrow \{\}$
 5: for annotator $a_k \in A$ do
 6:     $q \leftarrow$ location probability distribution of annotator $a_k$
 7:     Calculate spatio temporal distance
        $d_st(p(x_t, t), q(x_t, t))$
 8:     Distance $d \leftarrow \{\}$
 9:     for each activity $w_i$ do
10:         $d(w_i, w_j) \leftarrow$ activity-activity distance
11:         $d.insert(d(w_i, w_j))$
12:     end for
13:     maximum activity-activity distance $d_max \leftarrow$
        $max(d)$
14:     $Y_f(a_k, user) \leftarrow$ relationship weight
15:     $D(a_k, user) \leftarrow$ annotator-user distance
16:     $J_{a_k} \leftarrow$ reward index for annotator $a_k$
17:     $J[k] = J_{a_k})$
18:     Maximum reward $J_{max} = max(J)$
19:     $i \leftarrow$ index of $J_{max}$
20: end for
21: return annotator $a_i$
```

Experimental Evaluation

In this section we evaluate *SocialAnnotator* using real data traces and compare the performance of our model using different bandit algorithm. We also evaluate our classifier based on annotator We provide a description of our setup and dataset and data collection process in the following:

Setup

We collected activity data using wearable devices from 5 users over the course of 16 days. We used android smart watch Moto360 to collect the accelerometer data. We also collected the location information of the users using GPS which we only used for ground truth. We developed smartphone apps for both ios and android platforms using which the users can add correspondence (friend, spouse, roommate etc.). Users were asked to log the interaction, location and activity data using this platform. Users were asked to log not only the in person interactions but also virtual or remote (messaging, talking over the phone, interaction through social network etc.) interactions as well. Logging too much



(a) ***(b)*** ***(c)*** ***(d)***

Figure 2: *(a) shows the distribution of normalized reward with respect to number of queries. (b) shows per-annotator labeling time distribution. (c) represents the stack plot of percentage of correctly labeled instances of all the connected users for each user. (d) Precision, recall and f1-score of our base classifier using different settings.*

interaction can become a burden, so only the number of interaction was required to log. We established ground truth using these log information. In our experiment we monitored 5 daily activities - $\{$*eating, sleeping, phone calling, working, cooking*$\}$.

The sensor data are directly uploaded to our lab server from the wearable device and we preprocess(feature extraction, filtering, noise reduction etc.) the data in the server. As previously stated in our pipeline in Figure 1, we train a supervised classifier first to recognize the performed activity. We have used a simple decision tree based classifier. Initially after training our model with labeled instances we achieved an average accuracy of 77%. Even if we have achieved low accuracy compared to the existing literature, we are only concerned about investigating how efficient labeling can help to improve the performance of activity recognition model. We have For our budgeted multi-armed bandit, the reward and cost of each annotator are sampled from a beta distribution. The parameters of the these distributions are sampled from [1,5]. The budget of our framework is chosen from the set $\{200, 300, 400, 500, 1000\}$. We compare our annotator selection model with different contextual multi armed bandit algorithm - LinUCB, ϵ-Greedy, EXP4 and Random sampling. In case of Random sampling, the annotator is chosen at random in each iteration. All the contextual bandit algorithms are executed up to 300 iterations per user in this experiment.

Table 1: *Statistics of SocialAnnotator compared to other multi-armed bandit algorithm*

	$\bar{r}$	$\bar{c}$	$\bar{r}/\bar{c}$	% opt
Random	0.763	0.793	0.962	1.67
LinUCB	0.758	0.814	0.932	0.8
ϵ-Greedy	0.796	0.886	0.8984	1.32
EXP4	0.864	0.810	1.067	61.17
SocialAnnotator	0.913	0.267	3.419	73.42

Bandit Performance

In Table 1, we list and compare average rewards($\bar{r}$), average costs($\bar{c}$), average reward to cost ratio ($\bar{r}/\bar{c}$) and the percentage of time optimal annotator gets selected (% opt) of different bandit algorithms. From the statistics we see that LinUCB and ϵ-Greedy perform worst with respect EXP4 and our model. Both these algorithms are not meant for problems with budget constaints and as a result they do not take budget into consideration. Our model can achieve higher reward at lower cost contrast to other bandits which verifies that we are choosing optimal annotator at each step. In figure 2a the trend of average reward obtained at each step is shown. It is evident that our proposed algorithm outperforms the other bandit algorithm settings. More context information like detailed interaction, fine grained location information etc. might further improve the model.

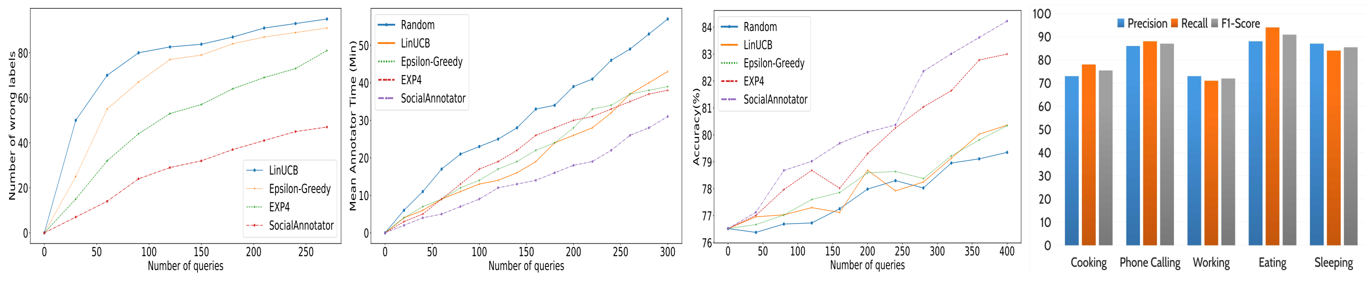
Annotator Selection

We monitor the performance of each annotator and maintain a score for each activity associated with the connected users. In Figure 2b we provide the annotation time distributions of each user using boxplot. A box depicts the majority of annotation times and the median time is marked with a solid line inside the box. It is noticeable from the figure that each user have different time distribution which means the efficiency, promptness and reliability of each user varies. We also deduce that the annotator might not provide the label at all. We show the percentage of correctly labeled instances by each user in Figure 2c. As *User 5* is only connected to *User 4* and *User 1* there are only three scores for him including the score of labeling his own activity. It is apparent that all the users are efficient in labeling their own activity.

Table 2: *Labeling result of each user*

	Correct Label	Wrong Label	No Label
User 1	324	49	27
User 2	306	68	26
User 3	285	83	32
User 4	310	73	17
User 5	345	37	18

We notice that *User 1* and *User 5* were able to label each others data quite precisely. We found that these two users were living in the same apartment and User 1 is *spouse* of User 5. Their quantity of interaction was also very high as apart from living together they were also talking with each other over the phone couple of times a day. We also notice from the figure that *User 2* and *User 4* were able to label the



(a) *(b)* *(c)* *(d)*

Figure 3: *(a) shows the number of wrong labels received in 300 iterations for different bandit algorithms. (b) per-annotator labeling time distribution (c) demonstrates the progression of accuracy after each iteration. (d) Precision, recall and f1-score of each activity.*

activity of each other with good accuracy (82% for *User 2* and 78% for *User 4*). After investigating into it, we observed that these two users were working together at the same place and had a lot of interactions. *User 3* and *User 4* also worked at the same place but they had very less interaction with each other which is reflected in their annotation efficiency. Receiving the label information as early as possible is aslo imperative. If we do not receive a label for the queried instance within a certain pre-defined threshold tme, we discard the annotator. As a result, if we pose the query to an user who may delay in providing the label or not provide the label at all, we not only spoil resources but also lose valuable information. In Figure 3b we show the mean annotation time needed using different bandit algorithm for varying number of queries. Our model exhibits lowest mean (30 mins) than all other approaches and getting the labels at the right time. As a result our model also ensures immediate result along with the conservation of information. To further validate our claim, we show the number of wrong labels received for different bandit algorithms in 300 iterations in Figure 3a. After 300 iterations, *SocialAnnotator* indicates lowest number of wrong labels, which proves that we are posing the queries to the right person at the right time. The cumulative labeling accuracy of each user is also described in Table 2.

Classifier Performance

We have achieved an overall accuracy of 77% for our base classifier. We trained our model with only 5% (1050) of the total labeled data instances. We apply active learning and incrementally query instances. We show the overall accuracy of our classifier after 400 iterations in Figure 2d. By employing *SocialAnnotator* we accomplish an average accuracy of $\approx$ 84% which is an improvement of 7% compared to our base classifier. *NoAnnotator* title demonstrates the results when we do not administer annotator selection. We notice that it only improve the accuracy by $\approx$ 1-2% even if we have applied active learning. So posing the query to the right person helps to improve the accuacy of our classifier. In Figure 3d we show the accuracy of individual activities after applying $SocialAnnotator$. We see that $Cooking$ and $Working$ show low accuracies with respect to other activities. After further investigation, we noticed that these two activities itself are very complex and experienced low accuracies in our base classifier as well (68% and 64%). However, *SocialAnnotator* actually increased the accuracy by $\approx$ 8-10%. Consequently $SocialAnnotator$ helps to improve the accuracies of complex activities which are hard to infer. Figure 3c shows the change of accuracy in 400 iterations. Our algorithm converges to optimum accuracy faster than other approaches.

Conclusion

In this paper we have proposed a novel annotator selection method $SocialAnnotator$, by exploiting social relationships among the users to improve the efficiency of active learning in activity recognition context. Our proposed model selects annotator based on the strength of the relationships and *spatio-temporal* distance metrics among the users. We also consider the similarities between the activities in our model to calculate the level of *correspondence* among the users. Prior works with active learning that propose to mitigate the labeling effort, have not considered the influence of annotators in their model. Our results show that, $SocialAnnotator$ can compliment active learning and establish reliabile, prompt and accurate label information. We have demonstrated that by using our methodology, we improved the accuracy of our base classifier by $\approx$ 7%. In our current approach while calculating the distance between two users, we only consider a very few interactions between them. In future we want to apprehend more interactions as well as more context information unobtrusively. We want to monitor the users phone usages and social network interactions without needing any feedback from them and add more sensor modalities like ambient infrastructure sensor to record the movements in detail. We also plan to do the regret anaylsis of our algorithm and derive the upperbound in future.

References

Bagaveyev, S., and Cook, D. J. 2014. Designing and evaluating active learning methods for activity recognition. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, 469–478.

Bhattacharya, S., and Lane, N. D. 2016. From smart to deep: Robust activity recognition on smartwatches using deep learning. In *IEEE International Conference on Pervasive Computing and Communication Workshops*.

Bhattacharyya, A. On a measure of divergence between two statistical populations defined by their probability distributions. *Bulletin of the Calcutta Mathematical Society*.

Bodó, Z.; Minier, Z.; and Csató, L. Active learning with clustering. In *Active Learning and Experimental Design workshop, In conjunction with AISTATS 2010*.

Bouchard, K.; Bouchard, B.; and Bouzouane, A. 2012. Unsupervised discovery of spatial relationships between objects for activity recognition inside smart home. In *The 2012 ACM Conference on Ubiquitous Computing*, 655–656.

Daiber, F., and Kosmalla, F. 2017. Tutorial on wearable computing in sports. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 65:1–65:4.

Davila, J. C.; Cretu, A.; and Zaremba, M. B. 2017. Wearable sensor data classification for human activity recognition based on an iterative learning framework. *Sensors*.

Diethe, T.; Twomey, N.; and Flach, P. 2016. Active transfer learning for activity recognition. In *24th European Symposium on Artificial Neural Networks, ESANN 2016*.

Donmez, P., and Carbonell, J. G. 2008. Proactive learning: Cost-sensitive active learning with multiple imperfect oracles. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM.

Fiorini, L.; Cavallo, F.; and et al. 2017. Unsupervised machine learning for developing personalised behaviour models using activity data. *Sensors*.

Gjoreski, H., and Roggen, D. 2017. Unsupervised online activity discovery using temporal behaviour assumption. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers, ISWC*.

Guan, Y., and Plötz, T. 2017. Ensembles of deep LSTM learners for activity recognition using wearables. *IMWUT*.

Hasan, M., and Roy-Chowdhury, A. K. 2015. Context aware active learning of activity recognition models. In *2015 IEEE International Conference on Computer Vision (ICCV)*.

He, Z.; Liu, Z.; Jin, L.; Zhen, L.-X.; and Huang, J.-C. 2008. Weightlessness feature x2014; a novel feature for single triaxial accelerometer based activity recognition. In *2008 19th International Conference on Pattern Recognition*.

Ho, Y.-c.; Lu, C.-h.; Chen, I.-h.; Huang, S.-s.; Wang, C.-y.; and Fu, L.-c. 2009. Active-learning assisted self-reconfigurable activity recognition in a dynamic environment. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1567–1572.

Hossain, H. M. S.; Khan, M. A. A. H.; and Roy, N. 2017a. Active learning enabled activity recognition. *Pervasive and Mobile Computing*.

Hossain, H. M. S.; Khan, M. A. A. H.; and Roy, N. 2017b. Soccermate: A personal soccer attribute profiler using wearables. In *IEEE International Conference on Pervasive Computing and Communications Workshops*.

Jalal, A.; Kim, Y.; Kim, Y.; Kamal, S.; and Kim, D. 2017. Robust human activity recognition from depth video using spatiotemporal multi-fused features. *Pattern Recognition*.

Lara, O. D., and Labrador, M. A. 2013. A survey on human activity recognition using wearable sensors. *IEEE Communications Surveys Tutorials*.

Lee, Y.-S., and Cho, S.-B. *Activity Recognition Using Hierarchical Hidden Markov Models on a Smartphone with 3D Accelerometer*.

Li, W., and Vasconcelos, N. 2017. Complex activity recognition via attribute dynamics. *International Journal of Computer Vision*.

Liu, R.; Chen, T.; and Huang, L. 2010. Research on human activity recognition based on active learning. In *2010 International Conference on Machine Learning and Cybernetics*, volume 1, 285–290.

Love, J., and Hirschheim, R. Crowdsourcing of information systems research. *EJIS*.

Münzner, S.; Schmidt, P.; Reiss, A.; Hanselmann, M.; Stiefelhagen, R.; and Dürichen, R. 2017. Cnn-based sensor fusion techniques for multimodal human activity recognition. In *Proceedings of the 2017 ACM International Symposium on Wearable Computers*, ISWC '17, 158–165.

Samarah, S.; al Zamil, M. G. H.; Al-Eroud, A. F.; Rawashdeh, M.; Alhamid, M. F.; and Alamri, A. 2017. An efficient activity recognition framework: Toward privacy-sensitive health data sensing. *IEEE Access* 5.

Shoaib, M.; Bosch, S.; Incel, Ö. D.; Scholten, H.; and Havinga, P. J. M. 2015. A survey of online activity recognition using mobile phones. *Sensors*.

Srba, I., and Bieliková, M. 2010. Tracing strength of relationships in social networks. In *Proceedings of the International Conference on Web Intelligence and International Conference on Intelligent Agent Technology*.

Stikic, M.; Van Laerhoven, K.; and Schiele, B. 2008. Exploring semi-supervised and active learning for activity recognition. In *Proceedings of the 2008 12th IEEE International Symposium on Wearable Computers*, ISWC '08, 81–88.

Twomey, N.; Diethe, T.; Craddock, I.; and et al. 2017. Unsupervised learning of sensor topologies for improving activity recognition in smart environments. *Neurocomputing*.

Wyatt, D.; Philipose, M.; and Choudhury, T. 2005. Unsupervised activity recognition using automatically mined common sense. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1*, AAAI'05, 21–27.

Xia, Y.; Qin, T.; and et al. Budgeted multi-armed bandits with multiple plays. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*.

Xu, J.; Xu, J. Y.; Song, L.; Pottie, G. J.; and van der Schaar, M. 2014. Context-driven online learning for activity classification in wireless health. In *2014 IEEE Global Communications Conference*, 2423–2428.

Yang, Q., and Wooldridge, M., eds. 2015. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*. AAAI Press.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

LAAIR: A Layered Architecture for Autonomous Interactive Robots

Yuqian Jiang*[1], Nick Walker*[2], Minkyu Kim[3], Nicolas Brissonneau[3], Daniel S. Brown[1], Justin W. Hart[1], Scott Niekum[1], Luis Sentis[3], Peter Stone[1]
[1]Department of Computer Science, University of Texas at Austin, Austin, USA
[2]Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, USA
[3]Department of Aerospace Engineering and Engineering Mechanics, University of Texas at Austin, Austin, USA
{jiangyuqian, nickswalker, steveminq, nicolasb}@utexas.edu
{dsbrown, hart, sniekum}@cs.utexas.edu, lsentis@austin.utexas.edu, pstone@cs.utexas.edu

Abstract

When developing general purpose robots, the overarching software architecture can greatly affect the ease of accomplishing various tasks. Initial efforts to create unified robot systems in the 1990s led to hybrid architectures, emphasizing a hierarchy in which deliberative plans direct the use of reactive skills. However, since that time there has been significant progress in the low-level skills available to robots, including manipulation and perception, making it newly feasible to accomplish many more tasks in real-world domains. There is thus renewed optimism that robots will be able to perform a wide array of tasks while maintaining responsiveness to human operators. However, the top layer in traditional hybrid architectures, designed to achieve long-term goals, can make it difficult to react quickly to human interactions during goal-driven execution. To mitigate this difficulty, we propose a novel architecture that supports such transitions by adding a top-level reactive module which has flexible access to both reactive skills and a deliberative control module. To validate this architecture, we present a case study of its application on a domestic service robot platform.

Introduction

Researchers have long sought to develop robots that are able to undertake complex tasks autonomously in real-world environments. Early efforts to develop such robots resulted in deliberative systems, in which the robot plans a sequence of actions to achieve a goal, and reactive systems, in which layers of local behaviors react to sensor input (Kortenkamp, Simmons, and Brugali 2016). Hybrid architectures, which layer deliberation and reactivity, emerged as a promising approach to the creation of integrated autonomous robots.

As efforts to improve individual robotic capabilities dominated the research landscape, the outline of hybrid architectures for general purpose robotics has seen few changes in the past two decades. In this time, researchers have made significant progress towards robust robot capabilities such as autonomous localization and navigation (Fox et al. 1999), object manipulation (Gualtieri et al. 2016), object recognition (Krizhevsky, Sutskever, and Hinton 2012). This increased capacity—the fruit of better system modeling, increased computational power, and new tools and techniques—has made it feasible for robots to act intelligently in more dynamic, real-world settings.

There is a renewed vision that robots will not only operate autonomously, but do so in challenging environments such as the home or office, where they must regularly interact with humans. Many projects have taken on the grand challenge of creating interactive autonomous robots for use in daily settings (Khandelwal et al. 2017), and international robotics communities have created competitions such as RoboCup@Home (Holz and Iocchi 2013) and the World Robot Summit Challenge[1] to encourage research efforts in this direction.

Though some hybrid architectures for general purpose robots were created with human interaction in mind, the extent to which robots operate in populated human environments was neither possible nor expressly accounted for in their design. In a typical layered architecture like 3T (Peter Bonasso et al. 1997), skills, which control the connections between sensors and actuators, are placed at the bottom. These low-level control components are invoked and monitored by the middle sequencer layer to achieve planned behaviors. At the top, a task planning layer decomposes the current task into a plan of lower-level behaviors. Given an input task specification, these layers provide the complete loop of planning, monitoring and executing a task.

This design does not address the desire for robot assistants that constantly interact with people and dynamically receive and execute all kinds of tasks in the environment over extended periods of time. In order to seamlessly respond to human interactions, the top layer must be reactive and maintain direct control of the lower level components, such as a dialogue handling skill and a component that parses commands.

In this paper, we propose a layered architecture for autonomous interactive robots, LAAIR, to facilitate complex tasks in long-term settings and dynamic interactions with humans in real-world domains. The top-level of a LAAIR system is reactive control, which sequences and executes skills in response to the environment and interactions. When the top-level encounters tasks that cannot be statically decomposed, it invokes a deliberative controller which plans and executes actions to accomplish the goal. The bottom

*Equal contribution.

[1]`http://worldrobotsummit.org/en/about/`

level consists of skills which interface with the world. We contrast LAAIR with existing robot architectures and present a case study of applying LAAIR on a mobile robot platform.

Related Work

The problem of structuring the software of an intelligent robotic system has long been pursued. Early architectures centered on robot planning systems, built around "sense-think-act" cycles. Shakey, for example, leveraged a STRIPS planner (Nilsson 1984). At the time of their development, computational limitations made complex modeling methodologies such as those used for inverse-kinematic motion planning an inaccessibly slow process, and technologies for perception such as object recognition lacked the maturity of modern systems. Partially as a response, reactive systems, which more closely coupled sensing and acting, emerged. In the Subsumption architecture, a well-known example, the robot's behavior is governed by a hierarchy of reactive layers in which control from higher levels subsumes that of lower levels (Brooks 1986).

Hybrid architectures draw together deliberative and reactive control, most commonly by placing a high level planner in control of various reactive components. In three layer architectures, this connection is mediated by an executive, commonly a hierarchical state machine, which orchestrates the particular low-level skills used to accomplish a plan (Gat 1997). Notable examples of such architectures include 3T (Peter Bonasso et al. 1997), TCA (Simmons 1994) and ATLANTIS (Gat 1992).

Our architecture shares many attributes with ATLANTIS. In both architectures, the task planner is only called by the executive control. In ATLANTIS, this design was driven by the need to support asynchronous calls to the slow planning process. Where ATLANTIS makes the planner the primary decision maker for the system, in LAAIR, the top layer is reactive control. This layer is responsible for decomposing tasks, either by invoking the deliberative control layer or by directly sequencing skills. This is driven by the need to encode both static and dynamic task decomposition and maintain responsiveness during long-term autonomous deployments.

In recent decades, many advancements have been made towards general purpose autonomous interactive robots. Robot Operating System (ROS) has emerged as a dominant software framework in the community, encouraging roboticists to think of their system as an agglomeration of standard, interchangeable components (Quigley et al. 2009). Many robotic systems have leveraged ROS and other standardized software components as a foundation on which to improve specific skills or tasks.

These efforts have laid the groundwork for new approaches to the design of general purpose robot architectures, the challenge LAAIR addresses. While several integrated systems have been designed to address particular challenges for service robots, such as interfaces to natural language commands (Chen, Yang, and Chen 2016), planning in realistic domains (Tran et al. 2017; Hanheide et al. 2017), and real-world scene perception (Beetz et al. 2015).

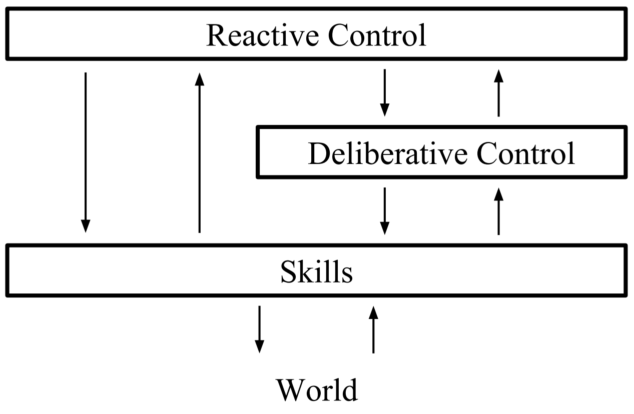


Figure 1: The prototype for a LAAIR system

Although these systems tackle similar challenges in architecting complex robotic systems, none of these efforts have proposed an overall architecture to support general purpose service robots. LAAIR was designed to organize and reuse interfaces and skills across robots and tasks.

LAAIR

LAAIR, depicted in Figure 1, is a three layer hybrid architecture, consisting of a pool of modular skills, a deliberative control layer that can sequence these skills to achieve goals, and a reactive control layer which drives the system's behavior. The reactive control layer can either directly schedule skills or delegate to the deliberative control layer. Each layer has asynchronous supervisory control over the layers beneath it. This makes it easy to specify systems that remain responsive to human interaction by, for instance, running a user engagement detection skill from the reactive layer, which can then redirect the rest of the system's behavior.

LAAIR facilitates code and logic reuse by separating low-level, often robot-specific implementation of skills from robot-independent task structure. The top-level reactive layer simplifies the specification of the top-level scripted behavior of the robot. The deliberative control layer provides the robot with the ability to reason about its environment flexibly as necessary.

Skills

Skills are the primary interface between the system and the world. They encode robot behaviors, ranging from low-level actions, like moving a joint, to higher level actions, like picking up an object. Everything from perceptual capabilities to the robot's dialog agent are implemented as skills. Skills may accept parameters and are responsible for detecting and reporting their own failure. Importantly, skills must adhere to some uniform interface so they can be directly sequenced by either reactive or deliberative control. This constraint promotes the reuse of skills in different contexts and the portability of the architecture across different platforms. Outside of these constraints, skills may be implemented as best suits their objective—whether it be a procedural program, a state machine or otherwise.

Deliberative Control

The deliberative control layer is responsible for turning a goal into a sequence of actions that accomplish that goal. These actions should either map directly to skills or be decomposable into a sequence of skills. In addition to the actual process of generating the sequence of skills, the deliberative control layer is responsible for monitoring the execution of the sequence, intervening when it determines that resequencing or other corrective action is necessary. The deliberative control layer reports major milestones in execution, such as completed actions or exceptional behavior to the reactive control layer, so that they may be optionally handled in a task-specific manner.

Because LAAIR gives the reactive control layer discretion on the specification of goals, the architecture can be instantiated with a broad range of deliberative components, or even use different components within the course of accomplishing a single task. Further, goal specification flexibility allows the reactive control layer to statically decompose a complicated task into several goals to limit the computational expense of deliberation.

Reactive Control

The reactive control layer is the primary executive in the system. It contains a high-level representation of the robot's task, for instance, a hierarchical finite state machine. In simple cases, this layer uses a static representation of the task to sequence skills and handle contingencies for different execution outcomes. In complicated tasks where static decomposition is infeasible, this layer produces a goal or goals that can be dynamically resolved by the deliberative control layer into a sequence of skills. The reactive control layer is responsible for monitoring and handling the outcomes of the skills that it directly calls, as well as for supervising deliberative execution. This gives the layer overarching control of the system, allowing it to preempt execution when, for instance, a human engages with the robot.

The reactive control layer also facilitates the hand-specification of actions for static parts of a task, while simultaneously supporting the use of deliberative control for instances where the task is dynamic. Because executive control over the rest of the system is maintained from this layer, the robot's behavior is always attributable to some portion of its top level representation. This makes LAAIR systems easier to understand at runtime and easier to debug during development.

Case Study

We show the instantiation of LAAIR on a Toyota Human Support Robot (HSR) as part of an entry into the 2018 RoboCup@Home Domestic Standard Platform League. The system is designed to execute both highly prescribed tasks, like guiding a person from a vehicle to an apartment, as well as open-ended tasks, like servicing requests that could require the robot to execute any number of actions in arbitrary order.

HSR is a domestic robot platform equipped with an omni-directional base, an arm, stereo and RGB-D vision systems, a speaker as well as a microphone array. The software stack of HSR is based on ROS. In this instantiation of the architecture, depicted in Figure 2, the reactive control layer is provided by hierarchical finite state machines, deliberative control leverages a symbolic task planner, and skills range from custom implementations to open source components.

Reactive Control

We use SMACH[2] to implement hierarchical finite state machines describing each RoboCup@Home task. Many of the tasks follow a static series of steps, so they are implemented purely by sequencing skills. When the robot is required to accept natural language commands and dynamically decompose the task, we enter a sub-state machine which calls skills to formulate a goal that can be delegated to the deliberative control layer. If the plan execution encounters errors, a sub-state machine in the reactive control layer engages to either specify another goal or execute a remedial sequence of skills. The reactive layer connects a command dialogue skill to a constantly-running wrist tap detection skill to enable the robot to accept new tasking.

Deliberative Control

We implement the deliberative control of our robot using a task planner and a plan executor. The task planner is based on Answer Set Programming (ASP) (Lifschitz 2002), and we use the answer set solver CLINGO to generate plans (Gebser et al. 2011). The plan executor is responsible for invoking the planner after receiving a goal, calling the corresponding skills, and monitoring the execution.

Knowledge Base

Besides the control layers, central to our system is a knowledge base module that stores a concept network and situated knowledge about the domain. We implement the knowledge base with a relational database managed by MySQL. The knowledge base represents entities and attributes. Each entity is assigned an ID, and can represent an abstract concept or a concrete object in the environment. Attributes describe properties and relations of entities.

The knowledge base is accessed by all layers of control in the architecture: the reactive executive writes and reads the identity of the operator and other task-level information; the deliberative control plans over domain knowledge; the skill components retrieve relevant information and update state as necessary.

Skills

We leverage the action server/client abstraction available in ROS to implement skills as processes which can be provided a goal, send feedback to a supervisor, accept cancellation requests and return outcomes. In exceptional cases, such as skills that return significant amounts of data, we implement skills as library functions to avoid interprocess communication overhead. Adopting this standardized interface for most skills simplifies how the deliberative or reactive layers interacts with them.

[2]`http://wiki.ros.org/smach`

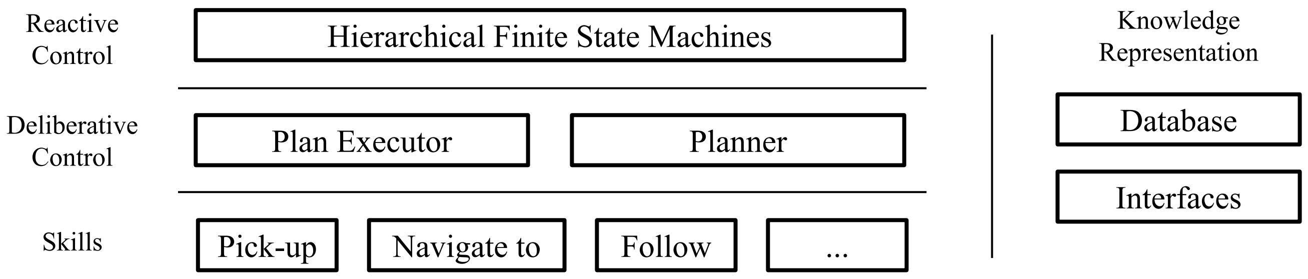


Figure 2: Functional components of our LAAIR implementation for HSR.

The selection of skills described in this section highlight the wide range of implementation in this layer.

Command Dialog agent This skill manages a dialogue with the operator, parses the command, and resolves the task type and parameters. The commands are generated from the fixed grammar used in the Robocup@Home Competition[3]. Our current system understands 32 high-level domestic tasks, such as finding people, answering questions, and delivering objects, which can be sequenced in any order to form a wide variety of complex commands.

Our implementation of the skill transcribes an utterance using the Google Cloud Speech-to-Text API. We then build a parse tree by expanding the production rules to all possible sentences. For each command, we traverse the parse tree to match the task type, such as "navigate to", and to fill in task parameters, such as "dining table".

After fully parsing a command, we resolve coreferences by searching backwards in the sentence to find the closest name or object. If a coreference cannot be resolved, or a command is incomplete, we engage in a correction dialogue to attempt to recover the missing information. Based on the semantics and type of missing information, the robot selects from a library of templates in order to ask an appropriate clarification question and resolve the parse.

Detection of movable objects We assume that objects which the robot may need to move—to clear a path for navigation, for instance—include a curved surface and at most two accumulated degrees of freedom. We detect potentially movable objects through two independent methods:

The first runs continuously, recording a 2D ground map of the environment based on laser and depth readings and comparing it with incoming readings. This method is robust to small changes to the environment, making it appropriate for detecting whether small items such as books, bags or cans on the floor have moved. The second searches for cylindrical surfaces in the scene and classifies them as part of potentially movable objects. This method is more efficient for identifying bigger obstacles such as rolling chairs or doors as it relies on their degrees of freedom.

Person Following We divide the person following skill into three steps; detecting the target, tracking the target with the robot's head, and navigating towards the target for following. These behaviors must be coordinated based on the state of the robot and its surroundings. We model this task as a behavior tree, which provides a compact representation of how the robot should move between actions while executing the behavior. Our implementation leverages a ROS behavior tree framework, described in (Colledanchise 2017). Our behavior tree is designed with two fallback nodes and two sequence nodes as shown in Figure 3.

[3]`https://github.com/kyordhel/GPSRCmdGen`

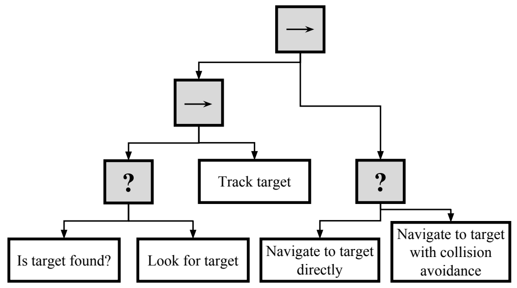

→
→
?
Track target
?
Is target found?
Look for target
Navigate to target directly
Navigate to target with collision avoidance

Figure 3: Behavior tree for person following skill. Arrows indicate sequence nodes and question marks indicate fallback nodes.

Future Work

Our case study has demonstrated the potential for LAAIR to address the needs of a general purpose robot operating in a human-populated environment. We are currently implementing LAAIR on an office robot platform to demonstrate how the architecture enables a high degree of software portability across service robots. In further instantiations, we plan to demonstrate the architecture's ability to handle interuptions and concurrency.

Acknowledgements

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF(IIS-1637736, IIS-1651089, IIS-1724157), Intel, Raytheon, and Lockheed Martin. Peter Stone serves on the Board of Directors of Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

Beetz, M.; Bálint-Benczédi, F.; Blodow, N.; Nyga, D.; Wiedemeyer, T.; and Marton, Z.-C. 2015. Robosherlock: Unstructured information processing for robot perception. In *IEEE International Conference on Robotics and Automation (ICRA-2015)*, 1549–1556. IEEE.

Brooks, R. A. 1986. A Robust Layered Control System For A Mobile Robot. *IEEE Journal on Robotics and Automation*.

Chen, K.; Yang, F.; and Chen, X. 2016. Planning with task-oriented knowledge acquisition for a service robot. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, 812–818. AAAI Press.

Colledanchise, M. 2017. *Behavior Trees in Robotics*. Ph.D. Dissertation, KTH Royal Institute of Technology.

Fox, D.; Burgard, W.; Dellaert, F.; and Thrun, S. 1999. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, 343–349. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Gat, E. 1992. Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-world Mobile Robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, 809–815. AAAI Press.

Gat, E. 1997. On Three-Layer Architectures. In Kortenkamp, D.; Bonnasso, R. P.; and Murphy, R., eds., *Artificial Intelligence and Mobile Robots*.

Gebser, M.; Kaufmann, B.; Kaminski, R.; Ostrowski, M.; Schaub, T.; and Schneider, M. 2011. Potassco: The potsdam answer set solving collection. *Ai Communications* 24(2):107–124.

Gualtieri, M.; ten Pas, A.; Saenko, K.; and Platt, R. 2016. High precision grasp pose detection in dense clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2016-Novem, 598–605. IEEE.

Hanheide, M.; Göbelbecker, M.; Horn, G. S.; Pronobis, A.; Sjöö, K.; Aydemir, A.; Jensfelt, P.; Gretton, C.; Dearden, R.; Janicek, M.; Zender, H.; Kruijff, G. J.; Hawes, N.; and Wyatt, J. L. 2017. Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence*.

Holz, D., and Iocchi, L. 2013. Benchmarking Intelligent Service Robots through Scientific Competitions : The RoboCup @ Home Approach. Technical report.

Khandelwal, P.; Zhang, S.; Sinapov, J.; Leonetti, M.; Thomason, J.; Yang, F.; Gori, I.; Svetlik, M.; Khante, P.; Lifschitz, V.; Aggarwal, J. K.; Mooney, R.; and Stone, P. 2017. BWIbots: A platform for bridging the gap between ai and human–robot interaction research. *The International Journal of Robotics Research*.

Kortenkamp, D.; Simmons, R.; and Brugali, D. 2016. Robotic Systems Architectures and Programming. In Bruno, S., and Oussama, K., eds., *Springer Handbook of Robotics*. Switzerland: Springer, Cham, 2 edition.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, 1097–1105. USA: Curran Associates Inc.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence* 138(1-2):39–54.

Nilsson, N. J. 1984. Shakey the robot. Technical Report 323, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025.

Peter Bonasso, R.; James Firby, R.; Gat, E.; Kortenkamp, D.; Miller, D. P.; and Slack, M. G. 1997. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence* 9(2-3):237–256.

Quigley, M.; Conley, K.; Gerkey, B. P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

Simmons, R. G. 1994. Structured control for autonomous robots. *IEEE transactions on robotics and automation* 10(1):34–43.

Tran, T. T.; Vaquero, T.; Nejat, G.; and Beck, J. C. 2017. Robots in retirement homes: Applying off-the-shelf planning and scheduling to a team of assistive robots. In *IJCAI International Joint Conference on Artificial Intelligence*.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Evaluating Predictive Knowledge

Alex Kearney, Anna Koop, Craig Sherstan, Johannes Günther
Richard S. Sutton, Patrick M. Pilarski, Matthew E. Taylor
Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada;
kearney@ualberta.ca

Abstract

Predictive Knowledge (PK) is a group of approaches to machine perception and knowledgability using large collections of predictions made online in real-time through interaction with the environment. Determining how well a collection of predictions captures the relevant dynamics of the environment remains an open challenge. In this paper, we introduce specifications for sensorimotor baselines and robustness-to-transfer metrics for evaluation of PK. We illustrate the use of these metrics by comparing variant architectures of General Value Function (GVF) networks.

Predictive Knowledge

A key challenge for machine intelligence is that of representation: a system's performance is tied to its ability to perceive and represent its environment. Predictive knowledge representations use large collections of predictions to model the environment. An agent continually anticipates its sensation from its environment by making many predictions about the dynamics of its environment with respect to its behaviour (Modayil, White, and Sutton 2014). These predictions about expected sensation can then be used to inform an agent's internal representation of its environment (Littman and Sutton 2002). Other proposals describe inter-relations of predictions, similar to TD Networks (Tanner and Sutton 2005; Makino and Takagi 2008) to enable abstract, conceptual representations by making predictions of predictions (Schapire and Rivest 1988).

In this paper we discuss the subtleties of evaluation predictive representation and propose two complimentary techniques. We specifically consider PK methods that 1) are able to expand their representations by proposing new predictions, 2) are able to self-verify their predictions through interaction with their environment, and 3) are able continually learn their predictions on-line.

To examine these evaluation metrics we use the General Value Function framework for predictive representations (White 2015). GVFs estimate the expected discounted return of a signal C defined as $G_t = \sum_{k=0}^{\infty}(\prod_{j=1}^{k}(\gamma_{t+j}))C_{t+k+1}$. Value is estimated with respect to a specific policy π, discount function γ, and cumulant c: $v(s;\pi,\gamma,c) = \mathbb{E}_\pi[G_t|S_t = s]$.

The parameters c, π, and γ are the *question parameters* which specify what a GVF is about; the *answer parameters*—such as the step size α and eligibility decay λ—describe how a learning method learns to answer the GVF question. GVFs can be learnt online, incrementally through methods such as Temporal-difference (TD) learning (Sutton 1988). The representational power of a given GVF network depends not just on the quality of the answers, but also in the architecture of the network, as illustrated in Figure 1.

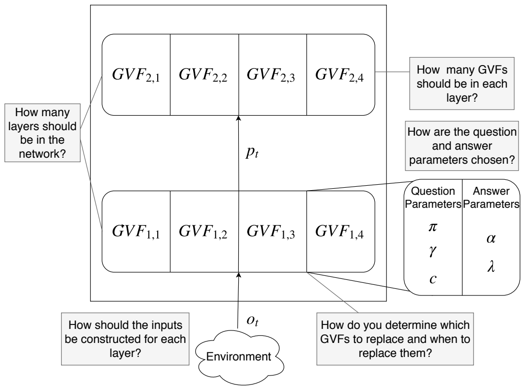

$GVF_{2,1}$
$GVF_{2,2}$
$GVF_{2,3}$
$GVF_{2,4}$
How many GVFs should be in each layer?
How many layers should be in the network?
How are the question and answer parameters chosen?
p_t
Question Parameters
Answer Parameters
π
γ
c
α
λ
$GVF_{1,1}$
$GVF_{1,2}$
$GVF_{1,3}$
$GVF_{1,4}$
o_t
How should the inputs be constructed for each layer?
Environment
How do you determine which GVFs to replace and when to replace them?

Figure 1: Many of the decisions which specify a PK architecture.

PK systems have been shown to be a scalable way to update and verify an agent's representation of the world, with examples of real-world robotic prediction tasks making thousands or tens of thousands of predictions in real-time on consumer-grade devices (Sutton et al. 2011; White 2015; Pilarski and Sherstan 2016).

Evaluating PK Architectures

Existing evaluation metrics for PK fall into two categories: 1) reporting the **average** error over all predictions within the PK system and 2) reporting errors on a known, challenging **subset** of the predictions within the system. Reporting the average error penalizes the accuracy of every prediction equally, when some predictions may have high error (such as for inherently random signals) but still provide representational power. Conversely, a representation that makes irrelevant but constant predictions will perform well according to average error, while providing no useful signals. Reporting errors on a subset of predictions requires identification of said subset across all architectures and lends itself to overfitting for those particular questions. It is difficult to identify predictions of interest without biasing towards particular architectures or network structures. Identifying predictions that require more complex representations in real-world settings requires extensive domain knowledge. In addition, it forces the inclusion of those pre-defined predictions when a goal of PK is to independently construct a useful representation. Neither of these are entirely satisfactory proxies for the real question: What is the representational power of a given PK system?

As a result of this evaluation bottleneck, examples of PK on real-world problems are largely proof-of-concept applications which serve to highlight the type, quantity, and diversity of predictions which can be made (Pilarski and Sherstan 2016; Modayil, White, and Sutton 2014; Sutton et al. 2011). Where evaluation exists, it focuses on prediction error as a means of evaluating the quality of a collection of predictions. This is insufficient, as *the reliability of predictions does not necessarily equate to the quality of a learned representation*. While low prediction error describes the quality of a single predictor, low average prediction error is not necessarily indicative of the best collection of predictions for constructing representations of the environment.

For example, one could maintain a diverse collection of GVFs for different time-scales γ and policies π that exclusively anticipate the voltage of servos on a robotic limb—a signal that is often constant. These trite predictions would likely have a lower error than a collection of predictions which represent the environment more completely. Moreover, comparing the average error between two collections of predictions with different question parameters is inappropriate, as the errors are with respect to different signals. When we compare the average error of different sets of predictions in PK architectures, we are unable to meaningfully quantify how changes in the architectural proposal impact the knowledgability of a system.

We propose sensorimotor predictions as a baseline which balances our ability to meaningfully assess the representational capacity of a collection of predictions in a meaningful way, while being general enough to be extensible to real-world prediction problems.

Evaluation by Sensorimotor Baselines

A scalable alternative to comparison by hand-crafted predictions is to maintain a collection of baseline sensorimotor predictions common between each architecture being evaluated. Instead of hand-crafting predictions based on the idiosyncrasies of a particular domain, a sensorimotor baseline uses the observations from the environment as prediction targets. The identification of good features is integral to being able to make reliable predictions; in evaluating the ability of a system to predict its raw stimuli, we are in fact evaluating the ability of the system to perform representation learning for the simplest predictions we could want to make.

By comparing architectures based on how well they can represent their stimuli, we are prioritizing architectures that are able to find better representations for learning low-level sensory input, rather than better representations of the environment in general. While a limitation, it is a natural approach to evaluation: approaches to PK have been motivated by being able to anticipate their environment (Modayil, White, and Sutton 2014), and low-level anticipatory predictions are useful as inputs in applications of PK (Sherstan, Modayil, and Pilarski 2015).

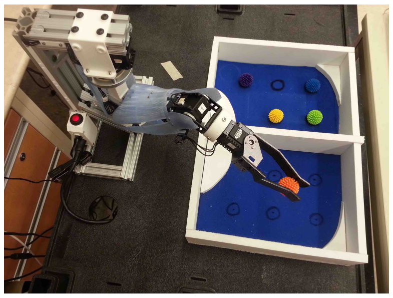


Figure 2: The data source for the experiments in this work: The Bento Arm, controlled by a human participant, generating a stream of multimodal sensory data from participants' interactions with a modified Box and Blocks task.

Sensorimotor baselines are a balance between the two aforementioned methods of evlauation: Baseline predictions enable us to assess the representation generated by our PK system with no designer intervention, making them a general scalable alternative for evaluation of real-world systems. By assessing representation quality, we can begin to precisely quantify the impact of different construction methods in real-world domains. Using sensorimotor baselines is a fair first step in bridging the evaluation gap between toy domains and real-world problems.

Evaluation by Transfer

Perhaps one of the most natural qualities of an effective PK system is generality. PK systems are intended for use in life-long, continual learning methods—methods that are expected to learn for the duration of their deployment. In such a setting it is imperative that the predictions being made are resilient to changes in their environment. A method of evaluating the ability of a continual learning system to produce general representations is through transfer-learning (Taylor and Stone 2009). We can evaluate the generality of PK by constructing GVFs in one setting and testing their generality on experience in a transfer environment that shares some traits with the source setting. An architecture that is able to propose and interrelate GVFs such that they are more robust to such transfers is an architecture that produces more general representations.

Experiment: Prosthetic Prediction Task

We explore sensorimotor baselines and transfer using data from a human control task on the Bento Arm (Dawson et al. 2014), an open-source robot arm intended for use as a research prosthesis. Human control of a robotic prosthesis is an area with active interest in PK (Pilarski and Sherstan 2016), and GVFs have been previously used to improve the control in this domain (Pilarski et al. 2013).

Data for this experiment was sourced from the previous experiments of (Edwards et al. 2016). Four users performed a common manipulation challenge where they used the robot arm to move objects over a barrier (Figure 2). Each user

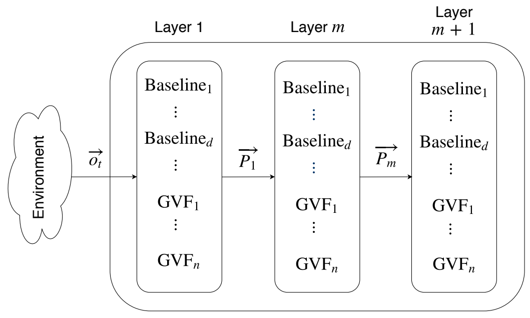


Figure 3: The architecture used for comparison. Each layer has d baseline predictions which predict each of the elements in the observations $\vec{o}$. Each layer has n additional predictions. The cumulants $c_{1...n}$ are functions of some output of the previous layer $\vec{p}_{m-1}$, or in the case of the first layer, the observations $\vec{o}_t$. For all predictions $\gamma = 0.95$, $\lambda = 0$, and step sizes are initialized to $\alpha_0 = \frac{1}{50}$, where 50 is the number of active features. Predictions are on-policy—π is always the robot arm's behaviour. Experiments vary the number of layers m and number of additional GVFs n.

performed the task three times using two different control schemes. We use one control scheme as the source environment and the other as a transfer environment, yielding 12 trials in total. The signals used to construct the observations are the position, load, velocity, and a binary movement signal for both the shoulder and hand joints.

The PK Architecture

To explore how choices in architecture impact the quality of learned predictions, we start with a straightforward representation using layers of GVFs (Figure 3). Inputs are produced in a feed-forward fashion: the base layer receives the observations from the environment o_t as state s_t, while each additional layer receives the output predictions from the previous layer. At each time-step, the position, velocity, and load of the shoulder and the gripper were used to construct the environment observations o_t. Step sizes are adapted using TIDBD (Kearney et al. 2017). We construct a binary representation of state by using a selective Kanerva coder (Travnik and Pilarski 2017) with 2000 prototypes and 50 active features.

In addition to the baseline sensorimotor predictions, there are n GVFs that are proposed and tested by the system. When proposing a new GVF, the architecture must specify both *what* the GVF is about by choosing c, γ, and π, and *how* the prediction is learnt by choosing appropriate learning parameters—in this instance, α and λ. Our architecture generates GVFs by randomly choosing cumulants, where c can either be an accumulation of a signal from the previous layer, or an operation on two signals—sums, differences, products, and ratios.

Each trial includes 20000 time-steps on a non-adaptive source setting where predictions are constructed, and 20000 time-steps on an adaptive switching transfer setting where the GVFs remain the same, but continue to be updated at each time-step. During the source setting, every 1000 time-steps the worst 10% of GVFs by average prediction error are culled and replaced with new GVFs, excluding the baseline

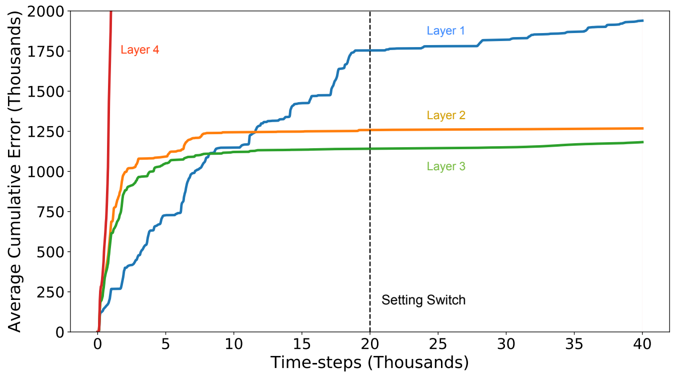


Figure 4: Accumulation of prediction error averaged over all sensorimotor baseline predictions in each layer. Our architecture has four layers and 100 constructed predictions in addition to the sensorimotor baseline. Error is averaged over 12 trials; variance is plotted but negligible.

predictions we use for evaluation. Prediction error is calculated online by estimating the discounted return from a sample of observed signals over approximately seven times $\frac{1}{1-\gamma}$, the expected time to termination.

Evaluation

To demonstrate evaluation using baseline predictions and setting transfer, we analyse the impact of two specification choices: 1) the number of layers in a network (Figure 4) and 2) the number of predictions in each layer (Figure 5). As indicated in Figure 1, these are decisions a designer must make when designing an architecture, and to date there is no clear intuition as to how these decisions impact the quality of the representation constructed. Our baseline predictions are of the load, position, velocity, and binary movement signal of the shoulder and gripper, or 8 predictions in total.

Since the first layer constructs its state exclusively from the observations from the setting o_t, it is not using any learned representations; by comparing each additional layer to the first layer, we assess how increasing representational abstraction impacts the baseline prediction error. Both the second and third-layer representations outperform predictions with no representation construction, while the fourth layer performs the worst. The sensorimotor baseline clearly illustrates the impact of abstraction on the ability to represent the environment.

There is a tension between what the GVFs in a layer can describe and the dimensionality of the representation for the following layer. Interestingly, the relationship between performance is not directly proportional to the number of predictions: while 10 additional predictions has the greatest performance, 100 additional predictions outperforms both 30 and 60 additional predictions. Of note is resilience to transfer to new prediction settings. Under none of the circumstances did the the methods accumulate substantially more baseline error after the switch to the transfer prediction setting. This demonstrates that the constructed representations generalized well between different control settings; however, in the future more complex transfer settings could be chosen.

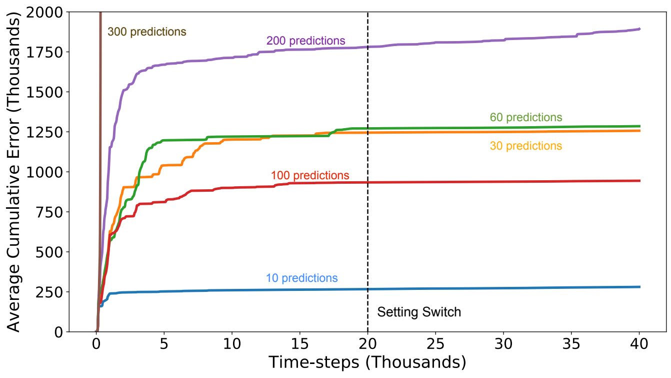


Figure 5: Accumulation of prediction error for averaged over all sensorimotor baselines predictions in the second layer. We vary the number of constructed predictions and the sensorimotor baseline. Error is averaged over 12 trials; variance is plotted but negligible.

By using a baseline of predictions, and performing transfer, we were able to elucidate how changes in the architecture impact predictive representations in a manner which requires little computational overhead. In doing so, we are providing a first step towards being able to study the impact of architectural choices on the learned representations of PK systems on real-world domains.

Limitations & Further Work

This paper's core contributions are a discussion of challenges in evaluation in PK and we do not perform an exhaustive evaluation of all the possible choices which could be made. For instance, we only consider the on-policy case, limiting the ability of our architecture to capture the impact of behaviour on the dynamics of the setting. In addition, future work could expand evaluation to include internal signals from predictions. For instance, internal signals of predictions corresponding to feature relevance could be used to identify the degree to which predictions used in the construction of state are impacting the model.

Conclusion

Predictive approaches to knowledge are a rich and varied area of reinforcement learning research that focus on building internal representations of the environment through continual, life-long interaction. There has been recent success in refining fundamental aspects of PK architectures on toy domains; however, these evaluation methods do not transfer effectively to large, real-world problems, such as applications in robotics, a core domain for predictive approaches to knowledge. In this paper, we highlight challenges in developing PK architectures and, as a primary contribution, propose the use of sensorimotor baselines and setting transfer to assess the quality of representations learned using PK. We demonstrate the usefulness of sensorimotor baselines and setting transfer by elucidating the impact of increasing the numbers of layers and number of predictions in each layer on the ability of an architecture to predict its stimuli. In providing preliminary evaluation methods for knowledge construction, we are taking a necessary step in the development of predictive knowledge.

References

[Dawson et al. 2014] Dawson, M. R.; Sherstan, C.; Carey, J. P.; Hebert, J. S.; and Pilarski, P. M. 2014. Development of the Bento Arm: An improved robotic arm for myoelectric training and research. *Proceedings of MEC* 14:60–64.

[Edwards et al. 2016] Edwards, A. L.; Dawson, M. R.; Hebert, J. S.; Sherstan, C.; Sutton, R. S.; Chan, K. M.; and Pilarski, P. M. 2016. Application of real-time machine learning to myoelectric prosthesis control: A case series in adaptive switching. *Prosthetics and orthotics international* 40(5):573–581.

[Kearney et al. 2017] Kearney, A.; Veeriah, V.; Travnik, J.; Sutton, R. S.; and Pilarski, P. M. 2017. Every step you take: Vectorized Adaptive Step sizes for Temporal Difference Learning.

[Littman and Sutton 2002] Littman, M. L., and Sutton, R. S. 2002. Predictive representations of state. In *Advances in neural information processing systems*, 1555–1561.

[Makino and Takagi 2008] Makino, T., and Takagi, T. 2008. Online discovery of temporal-difference networks. In *Proceedings of the 25th international conference on Machine learning*, 632–639. ACM.

[Modayil, White, and Sutton 2014] Modayil, J.; White, A.; and Sutton, R. S. 2014. Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior* 22(2):146–160.

[Pilarski and Sherstan 2016] Pilarski, P. M., and Sherstan, C. 2016. Steps toward knowledgeable neuroprostheses. In *Biomedical Robotics and Biomechatronics (BioRob), 2016 6th IEEE International Conference on*, 220–220. IEEE.

[Pilarski et al. 2013] Pilarski, P. M.; Dawson, M. R.; Degris, T.; Carey, J. P.; Chan, K. M.; Hebert, J. S.; and Sutton, R. S. 2013. Adaptive artificial limbs: A real-time approach to prediction and anticipation. *IEEE Robotics & Automation Magazine* 20(1):53–64.

[Schapire and Rivest 1988] Schapire, R. E., and Rivest, R. L. 1988. Diversity-based inference of finite automata. Master's thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science.

[Sherstan, Modayil, and Pilarski 2015] Sherstan, C.; Modayil, J.; and Pilarski, P. M. 2015. A collaborative approach to the simultaneous multi-joint control of a prosthetic arm. In *Rehabilitation Robotics (ICORR), 2015 IEEE International Conference on*, 13–18. IEEE.

[Sutton et al. 2011] Sutton, R. S.; Modayil, J.; Delp, M.; Degris, T.; Pilarski, P. M.; White, A.; and Precup, D. 2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *AAMAS 2011*, 761–768. International Foundation for Autonomous Agents and Multiagent Systems.

[Sutton 1988] Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3(1):9–44.

[Tanner and Sutton 2005] Tanner, B., and Sutton, R. S. 2005. Temporal-Difference Networks. In *International Conference on Machine Learning*.

[Taylor and Stone 2009] Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10(Jul):1633–1685.

[Travnik and Pilarski 2017] Travnik, J. B., and Pilarski, P. M. 2017. Representing high-dimensional data to intelligent prostheses and other wearable assistive robots: A first comparison of tile coding and selective Kanerva coding. *IEEE International Conference on Rehabilitation Robotics: [proceedings]* 2017:1443–1450.

[White 2015] White, A. 2015. *Developing a predictive approach to knowledge*. PhD Thesis, PhD thesis, University of Alberta.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

SOMA: A Framework for Understanding Change in Everyday Environments Using *Semantic Object Maps*

Lars Kunze[1], Hakan Karaoguz[2], Jay Young[3], Ferdian Jovan[4], John Folkesson[2], Patric Jensfelt[2], Nick Hawes[1]
[1]Oxford Robotics Institute, Dept. of Engineering Science, University of Oxford, United Kingdom
[2]Centre for Autonomous Systems, KTH Royal Institute of Technology, Sweden
[3]Sheffield Robotics, United Kingdom
[4]Howey Research Group, Dept. of Engineering Science, University of Oxford, United Kingdom

Abstract

Understanding change related to the dynamics of people and objects in everyday environments is a challenging problem. At the same time, it is a key requirement in many applications of autonomous mobile service robots.

In this paper we present a novel semantic mapping framework which maps locations of objects, regions of interest, and movements of people over time. Our aim with this framework is twofold: (1) we want to allow robots to reason semantically, spatially, and temporally about their environment, and (2) we want to enable researchers to investigate research questions in the context of long-term scenarios in dynamic environments.

Experimental results demonstrate the effectiveness of the framework which was deployed on mobile robot systems in real-world environments over several months.

1 Introduction

Everyday environments such as our homes, hospitals, and offices are dynamic and change over time. They change because people perform a broad range of everyday activities in them. During those activities, the locations of people and objects change as they move (or are moved) from one place to another. However, understanding change related to the dynamics of people and objects in everyday environments is a challenging problem. Yet it is a key requirement for autonomous robots to accomplish their tasks successfully.

For example, whilst *setting a table for dinner* a person gets plates, cups, knives, and forks from cupboards and drawers in the kitchen and lays them out on a dining table. However, robots can only make partial observations of those events due to their limited sensing capabilities and their egocentric viewpoints. Whilst observing such a scene, a mobile robot might only perceive a person moving through space and objects appearing and disappearing at various places at different points in time. As the robot only perceives snapshots of the underlying events it needs to reason about *what* it has seen, *where* and *when* to infer what to do next.

What have I seen? Where? and *When?* are essential questions in many robotic tasks such as searching for objects and the surveillance of human activities (Galindo and others 2008; Kostavelis and Gasteratos 2015). *Semantic environment maps* can provide answers to these questions as they link semantic information about the world (e.g. objects and people) to spatio-temporal representations. To this end, they are important resources in many robotic tasks. They allow autonomous robots to interpret (or ground) high-level task instructions; to plan and reason about how to achieve a task in a given environment; and to communicate observations and results to humans. However, constructing, maintaining, and using such maps in everyday, dynamic environments poses several challenges: (i) observations from sensor data need to be interpreted at a semantic level; (ii) interpretations need to be integrated into a consistent map; (iii) the map needs to be continuously updated to reflect the dynamic changes in an environment (over extended periods of time); and (iv) queries to the semantic map need to provide task-related information by taking semantic and/or spatio-temporal constraints into account.

In the past, many semantic mapping approaches in computer vision and robotics have addressed challenges (i) and (ii) by interpreting and integrating data from various sensors including laser rangefinders (Nuchter and Hertzberg 2008; Rusu and others 2009; Blodow and others 2011), stereo and monocular cameras (Sengupta and others 2013; Sunderhauf and others 2016), and RGB-D cameras (Pangercic and others 2012; Hermans, Floros, and Leibe 2014; Gunther and others 2015). These approaches assume that the environment is static and hence focus on the mapping of large-scale structures such as rooms, walls and furniture.

In this work, we address challenges (ii), (iii), and (iv). With respect to challenge (i), we use and adapt state-of-the-art robot perception methods that provide intermediate semantic interpretations from sensor data. Our work focuses on dynamic aspects of everyday environments including the varying locations of objects, regions of interest that potentially change over time, and movements of people. To this end, we investigate how objects, regions, and movements of people can be indexed by space and time so that map updates and queries can be handled both effectively and efficiently.

To address these challenges, we have designed, developed, and evaluated SOMA; a framework for constructing, maintaining, and querying *Semantic Object Maps*. In our work, *Semantic Object Maps* model semantic and spatial information about objects, regions, and trajectories of agents over time. Therefore they can provide answers to the questions: *What, Where* and *When?*

The presented framework allows autonomous robots to construct and update (or revise) maps automatically from

Table 1: SOMA concepts, their definition and examples.

Concept (Representation)	Definition	Examples
Object (3D pose & bounding box)	*Object* denotes a tangible thing that occupies a volume in space	Table, chair, monitor, cup, book
Region (2D polygon)	*Region* denotes an area of interest in space	Workplace, entrance, waiting area
Trajectory (2D pose array)	*Trajectory* denotes a path an agent has followed through space as a function of time	Human/ robot trajectory

observations using state-of-the-art perception methods and to query them from within the robots' control programs. At the same time, maps can be edited and queried by knowledge engineers and researchers for providing domain knowledge and for investigating research questions in long-term scenarios. It is important to note that we have designed SOMA with these two different user groups in mind: autonomous robots and researchers. While robots can add their observations and query maps for decision making, researchers can model aspects of the environment and/or analyse and extract spatio-temporal data collected by autonomous systems. The latter enables researchers to build and learn novel models about the (long-term) dynamics in everyday environments.

In this work, we have focused on the mapping for objects, regions, and humans in long-term and dynamic settings. Table 1 provides an overview of the high-level concepts used within SOMA. At an abstract level, our approach is similar to other works as it uses similar concepts for representing entities in the environment. Concepts such as objects, regions, and trajectories are natural and common sense. However, our approach differs significantly in the way we store, index, link, and query observations, interpretations, and semantic concepts over time. The main contributions of this work are as follows:

- an open-source semantic mapping framework (SOMA), designed for long-term, dynamic scenarios;
- a multi-layered knowledge representation architecture linking observations, interpretations, and semantic concepts using spatio-temporal indices;
- an adaptable mechanism for grounding objects in sensor data and a set of (extendable) interfaces for updating *Semantic Object Maps* over time;
- a query interface for retrieving and manipulating objects in *Semantic Object Maps* using semantic and spatio-temporal constraints; and
- a long-term case study of SOMA and *Semantic Object Maps* in a real world environment.

2 Related Work

Most research in the area of *robotic mapping* is based on metric representations (Thrun 2003). However, in the last decade, many approaches to *semantic mapping* have been proposed; a detailed account on the topic is given by (Pronobis and others 2010; Pronobis 2011) and an overview is provided by (Kostavelis and Gasteratos 2015).

(Capobianco and others 2016) propose a standardised way of representing and evaluating semantic maps. They define semantic mapping as an incremental process that maps relevant information of the world (i.e., spatial information, temporal events, agents and actions) to a formal description supported by a reasoning engine. Our work adopts a similar approach, we incrementally map spatio-temporal information about objects, people and regions and query those information using both standardised database queries and specialised inference mechanisms.

Several semantic mapping approaches have mainly focused on both the interpretation and the integration of data from various sensors including laser rangefinders, e.g. (Nuchter and Hertzberg 2008; Rusu and others 2009; Blodow and others 2011), stereo and monocular cameras, e.g. (Sengupta and others 2013; Sunderhauf and others 2016) and RGB-D cameras, e.g. (Pangercic and others 2012; Hermans, Floros, and Leibe 2014; Gunther and others 2015). Most of these approaches assume that the environment is static and hence focused on the mapping of static, large-scale structures such as rooms, walls and furniture. Our work is different from these approaches in two aspects. First, we do not develop methods for interpreting sensor data, but rather build on and adapt state-of-the-art robot perception methods (Aldoma and others 2012; Wohlkinger and others 2012), and secondly, we focus on the mapping, updating, and querying of semantic maps in *dynamic environments*.

A few semantic mapping methods approached the topic from a different angle. They focused on the design of ontologies and linking those to low-level environment representations (Zender and others 2008; Tenorth and others 2010). For example, work by (Pronobis and Jensfelt 2012) shows how different sensor modalities can be integrated with ontological reasoning capabilities to infer semantic room categories. Representing environment maps using Semantic Web technologies also enables robots to exchange information with other platforms via the cloud (Riazuelo and others 2015). We consider these types of approaches complementary to our work, as the semantic categories in our framework can be integrated and linked with exiting ontologies. For example, in (Young and others 2017a) hypotheses about objects are linked to structured, semantic knowledge bases such as DBpedia[1] and WordNet (Fellbaum 1998).

(Elfring and others 2013) presents a framework for probabilistically grounding objects in sensor data. In general, our framework does not make any strong assumptions about how objects are grounded in robot observations. Instead, the grounding of objects needs to be specified or learned by a user in the Interpretation layer of the framework (cf. Section 3.3).

[1] `http://wiki.dbpedia.org`

(Bastianelli and others 2013) present an on-line, interactive approach and an evaluation (Gemignani and others 2016) for the construction of semantic maps. Similarly, our work supports labelling of discovered objects by the crowd (cf. Section 3.5). However, our approach is off-line and is designed to work in an asynchronous way.

Our work is similar to (Mason and Marthi 2012) and (Herrero, Castaño, and Mozos 2015) which both focus on semantic querying of maps in dynamic environments. (Herrero, Castaño, and Mozos 2015) propose an approach based on relational databases which hold semantic information about objects, rooms and their relations required for mobile robot navigation. Our approach is similar as it also considers objects and regions in space (but not only rooms). However, in our approach relations between objects and regions do not have to be modelled explicitly, but can be inferred using spatial reasoning. (Mason and Marthi 2012) focus on semantic querying and change detection for objects. In their work, *objects* mean geometrically distinct occupied regions on a plane whose locations are described in a global reference frame. In contrast, our work can distinguish between unknown objects, classified objects, and known object instances. Our spatial indexing allows us to relate objects in a local, a global, and the robot reference frame. Furthermore, we can relate objects to regions and human trajectories.

Most similar to our approach is the semantic mapping framework by (Deeken, Wiemann, and Hertzberg 2018). Their framework is designed to maintain and analyse the spatial data of a multi-modal environment model. It uses a spatial database to store metric data and link it to semantic descriptions via semantic annotation. Spatial and semantic data can be queried from the framework to augment metric maps with topological and semantic information. This design and functionality is very similar to our approach. However, our approach goes beyond spatial and semantic information as it also includes temporal information about objects, regions, and people. Thereby it allows robots and users to reason not only about static configurations but also about temporally extended events such as everyday activities.

3 The SOMa Framework

3.1 Overview

Figure 1 provides a conceptual overview of the designed framework. The framework consists of two parts: (1) the SOMa core and (2) a set of SOMa extensions (or tools). Overall, the core has four layers. The three horizontal layers are interconnected and manage the information at different levels of abstractions: from observations (i.e. raw sensor data) and their interpretations to semantic concepts. These three layers are responsible for the representation within SOMa. The vertical interface layer provides access to all three levels. A set of extensions (or tools) use this layer for visualising, editing, querying and extending *Semantic Object Maps*. This allows knowledge engineers to extend and analyse them. Similarly, robots and user applications can access and manage maps through the interface layer.

Let us now consider the process of storing new information in SOMa. First, the robot's observations, in form of raw sensor data, are stored and spatially and temporally indexed in an observation layer. Second, an interpretation layer analyses these observations using perception methods, such as segmentation, object recognition, object classification, and people tracking, consolidates the results, and generates consistent descriptions at a semantic level. Finally, observations, interpretations, and semantic descriptions are linked together which allows the robot to query them at various levels using spatial, temporal, and/or semantic constraints.

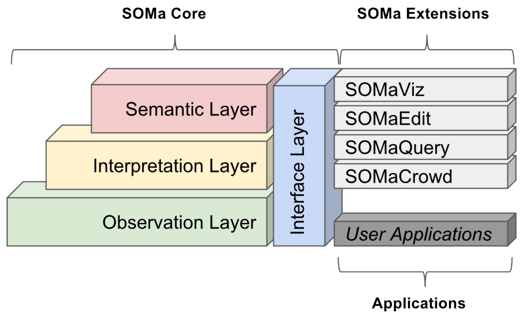


Figure 1: The SOMa framework consists of the SOMa Core and a set of extensions/tools for visualising, editing, and querying *Semantic Object Maps*.

3.2 Observation Layer

The role of the observation layer is to store both raw, unprocessed sensor data from the robot, along with any meta-data that might be useful in interpreting and processing that data by its systems. In order to accomplish this the observation layer stores the input from the robot's sensors during learning tasks. All other layers of SOMa also access this stored data. We store *views* which contain data from a single robot perception action, and we collect series of views into *episodes*. For our object learning tasks, a single *view* stores the point cloud, RGB image, depth image, current pose of the robot as well as any odometric transforms. An *episode* collects the series of views chosen by a planning algorithm for a particular learning task. Episodes and views may also have attached meta-data tags, which allows multiple different perception pipelines—perhaps all using different criteria to trigger, control and interpret data from learning tasks—to make use of the same database.

One of our design goals was to produce a way of storing raw robot perception data that would allow us to fully re-generate a SOMa database, performing all requisite processing steps along the way. Given just a copy of a robot's observation layer, this is possible, as the enclosed raw observations can be *re-played* as though they are being made in real-time. This is a key capability for evaluating different perception algorithms and pipelines on or off a robot. This functionality also helps in terms of fault tolerance—for instance, if a robot has been running for period of time with an undetected fault in a segmentation or object recognition layer, we are able to correct the error and fully re-generate the database from the observation layer, processing the data with the new corrected system, resulting in no loss of data.

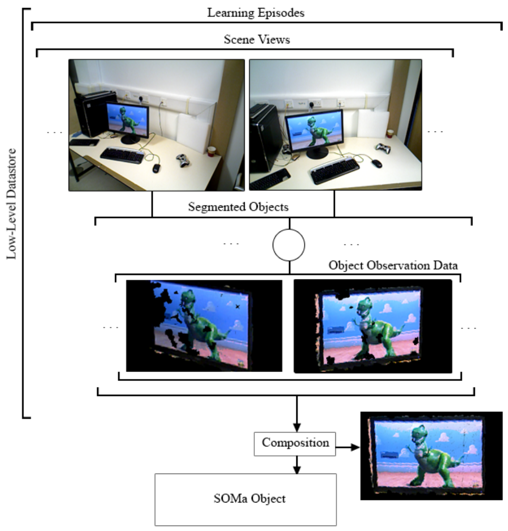
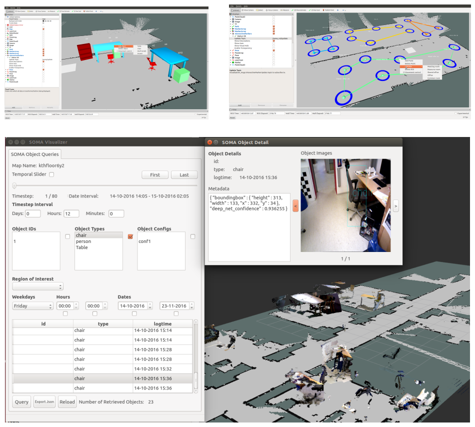


Figure 2: Data structures used in the Interpretation layer to compose high-level objects in the Semantic layer. First, individual views of a scene are stored. Second, object segments are found and linked with multiple observations of the same object across views. Finally, a SOMA object is created.

3.3 Interpretation Layer

The interpretation layer takes input from the observation layer and mainly contains application-specific methods for processing data. While the observation layer can be thought of as a wrapper around a robot's sensors, the interpretation layer is regarded as the part of the system that engages in application-specific processing of that data. In object learning, the first step in interpretation is to apply a segmentation algorithm, such as a depth-based segmentation or a region proposal network, in order to extract object proposals for further processing. SOMA provides a way of structuring the output of such segmentation algorithms by providing a scene graph-like object structure. This provides tools for storing data about individual segmented objects and their relationship to a view, and an episode, and allows a developer to collate observations of objects as further views are taken. The exact choice of algorithms used for scene segmentation, object tracking between views, and otherwise, are all left up to the developer as part of the design of their own application-specific interpretation layer.

Once the interpretation pipeline has processed and filtered the raw sensor output provided by the observation layer, high-level SOMA objects can be composed from the processed data. An example of this is shown in Figure 2. High-level objects represent the results of processing, and may record the output of object recognition algorithms over a series of views of an object, merged 3D models constructed from multiple views, and meta-data. These high-level objects link back to the low-level observations from which they are composed—the developer can go back and forth as desired from complete, merged objects to their constituent parts. The objects can then be used in further applications built on-top of SOMA—they can be shown to end-users in a



Map Name: kthfloor6y2
Temporal Slider
Timestep: 1 / 80 Date Interval: 14-10-2016 14:05 - 15-10-2016 02:05
Timestep Interval
Days: 0 Hours: 12 Minutes: 0
Object IDs: 1
Object Types: chair, person, Table
Object Configs: conf1
Region of Interest
Weekdays: Friday Hours: 00:00 00:00 Dates: 14-10-2016 23-11-2016

id	type	logtime
	chair	14-10-2016 15:14
	chair	14-10-2016 15:14
	chair	14-10-2016 15:28
	chair	14-10-2016 15:28
	chair	14-10-2016 15:32
	chair	14-10-2016 15:36
	chair	14-10-2016 15:36

Number of Retrieved Objects: 23

id:
type: chair
logtime: 14-10-2016 15:36

{ "boundingbox" : { "height" : 313, "width" : 133, "x" : 332, "y" : 34 }, "deep_net_confidence" : 0.936255 }

Figure 3: Examples of SOMA extensions. Top: SOMAEdit can be used for adding, editing or manipulating objects (left) and regions of interest (right). Bottom: SOMAViz is used for building spatio-temporal queries and visualising results.

labelling application, emailed, tweeted, visualised on a website, used in an application to locate lost mugs, processed further, or anything else the developer may desire.

3.4 Semantic Layer

The semantic layer stores high-level knowledge extracted from the robot's observations (cf. Table 1). The high-level knowledge could be the recognised object instances received from various recognition/detection pipelines or tracked objects from segmentation/tracking pipelines. Each high-level data instance is stored with spatio-temporal information such that the evolution of the knowledge about each object instance is maintained and can be retrieved. Moreover, each high-level SOMA object is linked with other SOMA layers by means of SOMA IDs in order to access all the knowledge about the object within the framework.

Furthermore, the semantic layer can store additional information about the object such as 3D models, camera images and any type of meta-data to build up a complete knowledge base. The stored high-level information could help the user to understand the semantics of individual environments, allow the robot to do high-level reasoning for accomplishing tasks such as object finding and/or grasping.

3.5 Interface Layer & SOMA Extensions

The interface layer acts as a backbone between the different SOMA layers and the user for exchanging data. As such, the robot/user can insert, delete, update and query data using SOMA extensions and other applications (Figure 3).

SOMAEdit allows users to create virtual scenes without any perceptual data. With this editor, users can add, remove or move objects and regions on top of a metric map.

SOMAQuery allows users to query maps using semantic, spatial, and/or temporal constraints. A query might ask for all objects of a certain type ("Select all cups"). Such a query can be further constraint by spatio-temporal constraints ("Select all cups in meeting rooms on Mondays be-

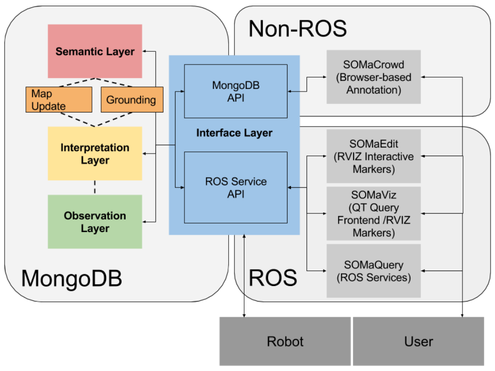


Figure 4: Implementation details of SOMA.

tween 10:00–12:00"). Spatial constraints can be used to determine if spatial entities are *near* to another entity, *within* an entity (region), or if they *intersect* with another entity. Temporal constraints can be formulated by using time points or temporal intervals. To discover temporal patterns and periodic processes *hour of the day*, *day of the week*, and *day of the month* are of particular interest.

Other extensions allow users to visualise query results (SOMAViz) and crowd-source missing object labels (SOMACrowd).

4 Implementation

We have implemented SOMA[2] based on ROS and MongoDB. The overall implementation structure of the framework is shown in Figure 4. ROS is used as the backbone of the entire SOMA framework as it is the most common platform used in the robotics research community. The individual SOMA layers and components are all developed as ROS nodes so that each of them can communicate with any other ROS component. Data structures used to store SOMA objects are themselves ROS messages composed of primitive ROS types. This provides a common interface between systems so long as they are built on the ROS stack.

For data storage and handling at all different layers, MongoDB is used. MongoDB provides great flexibility both in terms of the data structure and query capabilities. As such, all the SOMA data that is populated at different layers can be stored as ROS messages in MongoDB. Moreover, users can perform complex spatial queries using MongoDB's Geospatial API and/or projection criteria.

For visualisation tools, we have employed the Qt framework. Qt offers an end to end solution for developing graphical user interface as front-end and combining the back-end with ROS framework using multiple threads.

5 Experimental Validation

Our work was motivated by the European project, STRANDS (Hawes and others 2016). In STRANDS, we investigated spatio-temporal representations and activities in

[2]`https://github.com/strands-project/soma`

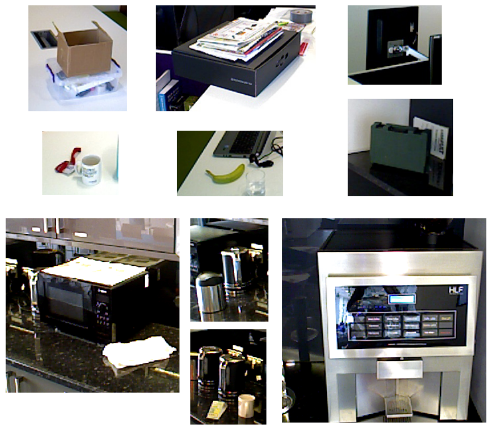


Figure 5: Objects discovered during deployment. SOMA stores the RGB-D images, point clouds, and other meta data.

long-term scenarios. Within the project, we were interested in providing services to humans in everyday environments. Tasks that our robots performed included object search, object discovery as well as activity recognition and movement analysis. In this context, we conducted a series of robot deployments in real-world office environments over several months in which we have evaluated this work.

Traditional, database-centric measures such as performance metrics over read/write access times and load-bearing tests would necessarily be evaluating the technology on which SOMA is *built*, rather than SOMA itself as a technology. Instead, we prefer to evaluate SOMA by looking at the range of applications it has been used for.

SOMA was used in multiple long-term deployments at two different sites—the Transport Systems Catapult (TSC) and a facility belonging to Group 4 Security (G4S), both in the United Kingdom. We report details of these deployments with respect to the three main entities that are represented within SOMA: *Objects*, *Regions*, and *Trajectories*.

5.1 Objects

While SOMA has been used as the underlying technology behind the development of more advanced and robust robot perception systems as described above, it has also served a dual role as the key component in many user-facing robot applications. Figure 5 shows a small sample of objects learned by the robot at the Transport Systems Catapult (TSC) deployment site, using the perception pipeline of (Young and others 2017b). In particular, 2D images of objects were used to pass to a Convolutional Neural Network (CNN) for identification, as in (Young and others 2017b; 2017a), as well as being passed to end-users at the site for live labelling. SOMA is also a data collection platform—we have used it to store and distribute scenes for future, offline labelling by human annotators. SOMA's performance in this area is dependent on the perception pipelines that feed information to it. Overall, the system stored 141 scenes and 341 scenes respectively, during the first and the second deployment at the TSC. The design of SOMA means that these

Table 2: Object learning performance (TSC, Y3/Y4).

Performance Measure	Y3	Y4
# Learning Episodes	56	**80**
# Views Taken	141	**341**
# Waypoints Visited	**25**	10
# Avg. Views / Learning Episode	∼2.5	**∼4**
# Avg. Episodes / Waypoint	∼2.24	**8**
# Autonomously Segmented Objects	445	**668**

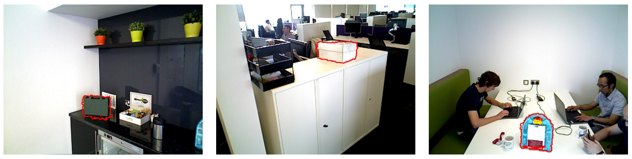


Figure 6: SOMA was used to generate reports of predefined surface areas in which objects were highlighted (TSC, Y3).

scenes can be re-processed later offline, using different perception pipelines, algorithms or filters if so desired, to extract different objects or different kinds of information from them. A comparison between the object learning pipelines used in year three (Y3) and year four (Y4) at the TSC site are shown in Table 2.

In the first deployment at TSC (Y3), SOMA was used to provide reports of objects discovered on predefined surfaces at the site (Figure 6). As the robot engaged in its normal object learning tasks, reports were generated and presented in a web-based blog interface for end-users to access.

In other experimental work, we designated two surfaces at the TSC site to be "learning tables", where office workers could bring objects for the robot to learn about. The robot would then visit the tables twice a day, and attempt to learn and identify any objects it had found. Using a CNN trained on a large image database, with 1000 possible categories, it would tweet about them while attempting to identify them. Internally, the system is made possible by SOMA's function of announcing when new objects are entered into the system, which then triggers the object identification and tweeting process by passing to those functions the 2D images of objects entered into SOMA.

During the last long-term deployment in TSC site (Y4),

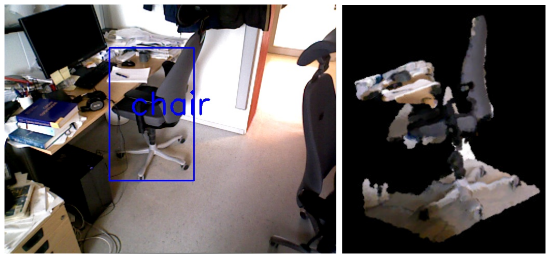


Figure 7: Processing steps of CNN-based object detection. Left: Detect object candidate. Right: Extract partial view.

Table 3: Detection results (TSC, Y4, ≈120 days).

Object type	Number of detections
People	178
Chairs	171
Monitors	104
Other objects	574

Table 4: Perceived objects (#) per time of day (TSC, Y4).

Hours	07:00-12:00	12:00-17:00	17:00-00:00
Monday	56	98	0
Tuesday	21	85	0
Wednesday	18	175	3
Thursday	184	224	0
Friday	0	67	0

we have employed a CNN based object detection pipeline. This pipeline was able to detect 20 object categories including person, chair, monitor, etc. and it was possible to extract a partial 3D view of the object using the registered depth information. As such the object location with respect to the robot and the global metric map can be identified. Figure 7 shows an example of a detected chair and the extracted partial 3D view. The detected objects were then stored as high-level SOMA objects with spatio-temporal information. Table 3 shows some detailed statistics about the objects detected with this pipeline during the deployment. The results show that the most detected objects were chairs, people and monitors which can be expected given that the robot was working on an office environment (Table 3).

We have also analysed temporal aspects of the Y4 deployment in terms of high-level SOMA object perception using the SOMAQuery interface. Table 4 shows the daily object perception statistics w.r.t the time of the day for the entire deployment. From the table it is observed that the robot was mostly active during Wednesdays and Thursdays while it has never been used in the weekends for object perception. It is also observed that the robot was most active during the afternoon hours but it was only rarely used during out of office hours (after 17:00). In total, the robot has perceived 930 high level SOMA objects during the entire Y4 deployment.

5.2 Regions

SOMA was used as the main memory component in (Karaoguz and others 2017) for human-centric partitioning of the environment. In the work, it was assumed that the co-occurrence of objects and humans can be used to identify densely populated areas. For this task, a CNN-based object detector and RANSAC-based tabletop detector were employed to detect objects. During the robot's operation, the detected objects were all stored as high level objects within in SOMA. After a set of observations were made, a reasoning module was employed to query SOMA objects and locate object clusters. These object clusters were then used to identify the dense regions. Figure 8 (left) shows the resulting regions. The proposed system discovered 16 regions of

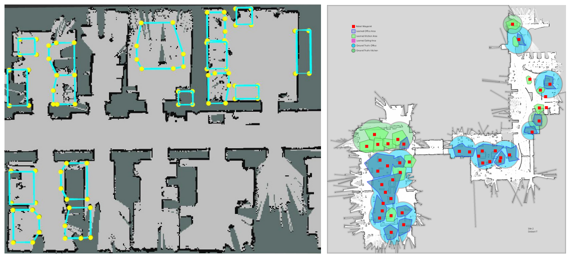


Figure 8: Examples of learned regions. Left: Auto-generated human-centric partitions of offices. Right: Learning of semantic labels associated with regions at TSC (Y4).

which 14 actually corresponded to manually annotated ones. The people density analysis showed that nearly 90% of the dense areas lay within or around the generated regions.

SOMA was also used in (Young and others 2017a) as part of a system for learning semantic labels associated with regions of space. Here, SOMA was used to represent objects discovered in the environment (Ambruş and others 2014). These objects were then passed to a CNN to be labelled. The system then used text mining of large text corpora (in this case Wikipedia) to find those room categories most strongly related to the labels of the discovered objects. Results from the TSC deployment are show in Figure 8 (right).

Discovery of these semantic room labels allowed us to draw bounding polygons around the areas of space where the related objects were observed. The result, as shown in Figure 8 where blue regions indicate office areas and green regions indicate kitchen areas, largely covered the same areas as annotated by human annotators. These learned, labelled regions can then be fed back in to SOMA and its own internal region representation, and potentially used by a robot for various tasks such as object search or activity recognition.

In general, SOMA has been a powerful API for improving both the internal object perception pipelines used on our robots—for instance, the region representation is key to our approach to view planning—but also a tool for building user-facing applications that provide a robot's-eye view of the world. SOMA's ability to support input from multiple, arbitrary perception pipelines has also been a great tool in development and debugging of these systems along with the suite of visualisation tools available.

5.3 Trajectories

In STRANDS, we have used a multi-sensor-based approach for people detection and tracking (Dondrup and others 2015). During the deployments, we gathered thousands of human trajectories at both sites. (Jovan and others 2016) analysed and predicted the level of activities at G4S. The level of activity was measured by the number of trajectories, within a particular time and location. This kind of analysis on human trajectories in temporal and spatial scale are possible via SOMAQuery. Temporal queries which involve periodic intervals during the week such as "Select all trajectories between 08:00–17:00 on every Monday" or specific time intervals such as "Select all trajectories between 08:00–12:00 on Sunday 12th March 2016" can be interpreted by SOMAQuery and visualised by SOMAViz. With help of SOMAQuery and background knowledge about the normative behaviour of people at G4S (e.g. "no employee is allowed to work during the weekend"), (Jovan and others 2016) employed a trajectory filtering to filter false detections captured during the deployment. A trajectory statistics for a set of temporal queries is shown in Table 5.

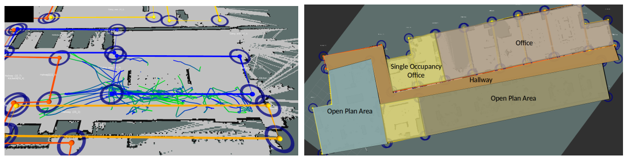


Figure 9: Left: Trajectories in kitchen area (TSC, Y4). Right: Region classification based on trajectories (G4S, Y2).

We used SOMAEdit to segment the sites' map into regions based on their functions. Given these regions, querying trajectories within these particular areas becomes possible. Thereby, SOMAQuery is able to interpret spatial queries such as "Select all trajectories within kitchen area" at different times of day (Figure 9). A brief comparison between G4S and TSC on spatial queries is shown in Table 5. (Jovan and others 2016) discovered that the temporal predictive model of human activities for each region provides a spatio-temporal signature which can further be used to classify regions based on their functionality. The region classification based on trajectories can be seen in Figure 9 (right).

6 Conclusions

In this paper, we have presented a semantic mapping framework for mobile robots, called SOMA. SOMA uses a three layered architecture to model objects, regions, and trajectories of agents. We explained how these layers interact with each other, how they can be accessed via an interface layer from both robots and user applications. We presented an experimental validation of SOMA by showcasing several use cases for the framework in real-world, long-term scenarios.

SOMA stores discrete observations of objects and regions. However, many new semantic mapping applications are based on continuous observations from sensors (e.g. cameras), and recent work on visual SLAM makes collection of this kind of data easy (Stückler and others 2015). We will investigate, in future work, how this kind of data, as well as segmented contents, can be integrated into our maps. Conceptually, however, our representation tools are general enough to support these, and other, new approaches. Hence, we believe that the open-source framework SOMA can have a wide within the robotics community.

Acknowledgments The work was supported by RAIN, EPSRC [EP/R026084/1]; Mobile Robotics Programme Grant, EPSRC [EP/M019918/1]; ALOOF, EPSRC [EP/M015777/1]; STRANDS, EU FP7 [600623].

References

Aldoma, A., et al. 2012. Tutorial: Point cloud library: Three-dimensional object recognition and 6 dof pose estimation.

Table 5: Trajectory statistics for temporal and spatial queries

Constraint Type	Query	G4S		TSC (Y4)	
		# Trajectories/hour	$\overline{\text{Length}}$	# Trajectories/hour	$\overline{\text{Length}}$
Temporal	Workdays 08:00-12:00	16.08	2.47	9.07	3.61
	Workdays 12:00-17:00	11.42	2.51	11.34	3.38
	Workdays 17:00-00:00	0.19	2.31	0.36	3.34
	Weekend	0.08	0.77	0.00	0.00
Spatial	Open Plan Area	3.02	2.45	0.18	3.90
	Kitchen	0.55	2.82	1.43	3.60
	Meeting Room	0.27	1.66	0.07	3.88
	Hallway	3.51	2.72	1.60	3.86

IEEE RAM 19(3).

Ambruş, R., et al. 2014. Meta-rooms: Building and maintaining long term spatial models in a dynamic world. In *IROS*.

Bastianelli, E., et al. 2013. On-line semantic mapping. In *ICAR*.

Blodow, N., et al. 2011. Autonomous semantic mapping for robots performing everyday manipulation tasks in kitchen environments. In *IROS*.

Capobianco, R., et al. 2016. A proposal for semantic map representation and evaluation. *CoRR* abs/1606.03719.

Deeken, H.; Wiemann, T.; and Hertzberg, J. 2018. Grounding semantic maps in spatial databases. *RAS* 105:146 – 165.

Dondrup, C., et al. 2015. Real-time multisensor people tracking for human-robot spatial interaction. In *ICRA*.

Elfring, J., et al. 2013. Semantic world modeling using probabilistic multiple hypothesis anchoring. *RAS* 61(2).

Fellbaum, C., ed. 1998. *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication.

Galindo, C., et al. 2008. Robot task planning using semantic maps. *RAS* 56(11).

Gemignani, G., et al. 2016. Interactive semantic mapping: Experimental evaluation. In *Experimental Robotics*.

Gunther, M., et al. 2015. Model-based furniture recognition for building semantic object maps. *Artificial Intelligence*.

Hawes, N., et al. 2016. The STRANDS project: Long-term autonomy in everyday environments. *IEEE RAM*.

Hermans, A.; Floros, G.; and Leibe, B. 2014. Dense 3D Semantic Mapping of Indoor Scenes from RGB-D Images. In *ICRA*.

Herrero, J. C.; Castaño, R. B.; and Mozos, O. M. 2015. An inferring semantic system based on relational models for mobile robotics. In *IEEE ICARSC*.

Jovan, F., et al. 2016. A Poisson-Spectral Model for Modelling the Spatio-Temporal Patterns in Human Data Observed by a Robot. In *IROS*.

Karaoguz, H., et al. 2017. Human-centric partitioning of the environment. In *ROMAN*.

Kostavelis, I., and Gasteratos, A. 2015. Semantic mapping for mobile robotics tasks: A survey. *RAS* 66:86 – 103.

Mason, J., and Marthi, B. 2012. An object-based semantic world model for long-term change detection and semantic querying. In *IROS*.

Nuchter, A., and Hertzberg, J. 2008. Towards semantic maps for mobile robots. *RAS* 56(11):915 – 926.

Pangercic, D., et al. 2012. Semantic object maps for robotic housework - representation, acquisition and use. In *IROS*.

Pronobis, A., and Jensfelt, P. 2012. Large-scale semantic mapping and reasoning with heterogeneous modalities. In *ICRA*.

Pronobis, A., et al. 2010. Semantic modelling of space. In *Cognitive Systems*.

Pronobis, A. 2011. *Semantic Mapping with Mobile Robots*. Ph.D. Dissertation, KTH Royal Inst. of Tech., Sweden.

Riazuelo, L., et al. 2015. Roboearth semantic mapping: A cloud enabled knowledge-based approach. *IEEE T-ASE* 12(2).

Rusu, R. B., et al. 2009. Model-based and learned semantic object labeling in 3d point cloud maps of kitchen environments. In *IROS*.

Sengupta, S., et al. 2013. Urban 3d semantic modelling using stereo vision. In *ICRA*.

Stückler, J., et al. 2015. Dense real-time mapping of object-class semantics from rgb-d video. *JRTIP* 10(4).

Sunderhauf, N., et al. 2016. Place categorization and semantic mapping on a mobile robot. In *ICRA*.

Tenorth, M., et al. 2010. Knowrob-map - knowledge-linked semantic object maps. In *IEEE-RAS Humanoids*.

Thrun, S. 2003. Exploring artificial intelligence in the new millennium. In *Robotic Mapping: A Survey*.

Wohlkinger, W., et al. 2012. 3dnet: Large-scale object class recognition from cad models. In *ICRA*.

Young, J., et al. 2017a. Making sense of indoor spaces using semantic web mining and situated robot perception. In *The Semantic Web: ESWC 2017 Satellite Events*.

Young, J., et al. 2017b. Semantic web-mining and deep vision for lifelong object discovery. In *ICRA*.

Zender, H., et al. 2008. Conceptual spatial representations for indoor mobile robots. *RAS* 56(6).

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

A Practical Distributed Knowledge-Based Reasoning and Decision-Theoretic Planning for Multi-robot Service Systems

A.-I Mouaddib and laurent Jeanpierre
GREYC–University of Caen Normandy
abdel-illah.mouaddib, laurent.jeanpierre@unicaen.fr

Abstract

This paper presents a practical model of distributed reasoning and planning for a fleet of robots serving people in a shopping mall using distributed knowledge-based reasoning and distributed Markov Decision Process (MDP) where the environment changes frequently and the set of goals is not static. The model we present, in this paper, consists of distributed local reasoning and planning where each robot locally reasons on its perceived data (locally: onboard cameras and also from global perception system: external cameras) to update its local Knowledge Base (KB). Local KBs derive local goals and the local planners select the goal to accomplish and compute the policy to accomplish it while maintaining a coordinated behavior with the other robots by avoiding conflicts on goals. To this end, we propose a distributed market-based auction planning algorithm using a regret and opportunity costs in a distributed value function leading to augmented MDPs to coordinate the robots and to select the appropriate goals to accomplish. Our approach assumes communication between robots and external sensor and we will describe a method to minimize the dis-coordination (conflits on goals) when the communication is lacking. Experimental results on the algorithm performance and the implementation on real service robots in a shopping mall showed a very satisfying behavior as shown in the video.

1 Introduction

The claim of this paper is to present a practical model and algorithm for a multi-robot system to be deployed in a public area like a shopping mall, airport, train station, hospital, ... to assist and guide customer. In our case, we consider assistance, advertisement and security goals in a shopping mall. The characteristics of such applications are :

- The environment is highly dynamic and a long-term horizon planning could be unsuitable.
- No central planner or system could be considered.
- The set of goals is dynamic and can change during the execution.
- The nature of the application is highly distributed in terms of perception, reasoning and planning since the robots sweep as large as possible the public space to detect events to handle.
- Goals accomplishments are durative and planning for the full set of goals could be inappropriate since new high prior goals could be generated.

Reasoning and planning in such contexts make some classical approaches inappropriate. Indeed, local KB reasoners of different robots can derive different goals distributed among their Knowledge-Base and central planner should compute a plan among all distributed goals to derive an optimal goal allocation. While most central planners require a central Knowledge base [Ghallab, Nau, and Traverso, 2016], planning with distributed knowledge bases becomes out of reach of these existing algorithms [Ghallab, Nau, and Traverso, 2016]. Formalizing the planning problem where robots have their own local observations leads to some strong mathematical tools like DEC-POMDP [Bernstein, Zilberstein, and Immerman, 2000; Amato, Bernstein, and Zilberstein, 2007; Seuken and Zilberstein, 2007; Dibangoye et al., 2016]. However, DEC-POMDPs require a central planner which make their use in this context inappropriate. Considering only interactions between agents to formalize the planning problem has been considered in different POMDPs-based approaches like Networked POMDPs [Nair et al., 2003; 2005], interactive POMPDs [Sonu and Doshi, 2015] or Augmented MDPs [Matignon, Jeanpierre, and Mouaddib, 2012]. Such approaches are promising to deal with distributed planning which is more appropriate to the problems where knowledge-based are distributed. Hiowever, they can show some limits where the environment is highly dynamic and the set of goals changes frequently. To this end, we extend the approach presented in [Iocchi et al., 2016] to the multi-robot system settings where the architecture is fully decentralized and combine auctionning and MDPs to coordinate the robot policies. The combination between auctionning and (PO)MDPs is not novel and has been introduced in [Spaan, Gonçalves, and Sequeira, 2010]. However the auctionning phase is centralized while our approach is fully decentralized and the auctionning is distributed among robots.

We present an approach which allows the robots to plan in a distributed way and with a changing set of goals using a distributed market-based auction algorithm combined with a distributed value function [Matignon, Jeanpierre, and Mouaddib, 2012] allowing each robot to locally plan and select the goals to accomplish. This algorithm has been implemented and tested on real robots serving people and guiding them to their requested destination showing a very satisfying behavior as shown in the video[1].

The rest of the paper is organized as follow: section 2 describes the overall architecture and the interaction between local reasoners and planners. Section 3 presents the distributed knowledge-base reasoning and the communication with the planners. Section 4 representing the main contribution of the paper where we describe the market-based auction algorithm and the distributed value functions using opportunity cost and the regret functions. Section 5 is dedicated to the evaluation of our approach in terms of solution quality and the implementation on real robots.

2 Distributed multi-robot Architecture

The distributed architecture of the multi-robot system consists of, as depicted in Figure 1, communicative local knowlege-based reasoners, planners and executors. Each robot has each local knowledge-based module which is splitted into two parts : static part describing the semantic map of the environment (shops, restaurants, social areas in the mall for instance) and the dynamic part describing the new incoming information from sensors (onboard cameras of the robots and external cameras in the environment) and from the other modules such as the status of the robots (idle or active), their goals under accomplishment and their priorities maintaining a global partial view on the overall system. The KB is a set of logical predicates similar to the classical STRIPS-like planning language [Ghallab, Nau, and Traverso, 2016]. The KB uses a simple rule-based reasoning to derive new goals to accomplish. These new goals are communicated to the planning module which uses a maket-based auctionning algorithm which is descibed bellow. This algorithm allows the robot to select the goal to accomplish and to formalize it as an MDP as presented in [Iocchi et al., 2016] and then communicate the policy to the executor module to act. In order to compute the goal to accomplish, each robot communicates with the others the set of locally derived goals and the vector of associated optimal policy values. Each robot fuses the set of received goals with the locally derived ones. This shared information allows to each robot to use global information to plan.

3 Distributed Knowledge-based reasoning

3.1 Local KB reasoning

Each robot maintains a local KB which is split into a static KB representing the semantic map and a dynamic KB representing the events occurring in the environments or on the status of the robot. Indeed, when external or on-board sensor detect an event (a person, an object for example), the KB inserts a logical formula representing this event and executes

[1]https://youtu.be/iFC6-sCL3XI

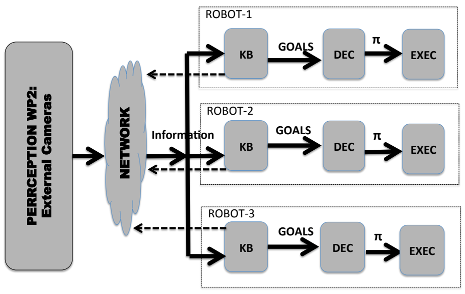


Figure 1: General multi-robot decision-making Architecture

its logical inference engine to derive new goals. These new goals are added to the existing ones and sorted according to their priority. In the shopping mall case, the security goals have higher priority than assistance goals which have high priority than advertising goals. This order allows the robots to start by allocating the goals according to their priority. The other knowledge in the dynamic part concern the information on the other robots particularly their status describing whether the robot is active and which goal is achieving or idle and its initial location when starting the execution of the goal achievement policy.

3.2 KB update and Synchronization

The local KB for each robot should be updated by external information coming from the environments (perception) and the other robots. In Figure 2, we represent information coming from the other robots by G_i^* meaning the goal under accomplishment by the other robot and the information coming from perception allow the KB module to generate G^t a goal list at time t. These information allow the KB module to generate the new list of goals: $G^t = G^t - G_i^*$. This list is then sent to the decision module which uses a distributed matrix-based auctioning algorithm, described in the next section, to select a goal to accomplish and computes the policy to accomplish it. The decision module updates the KB with the selected goal and the exec module about the policy to execute. The exec module during the execution sends the status of the execution and the current level of priority which is necessary for considering new goals or not. The exec module updates the KB at the end of the execution and this update is sent to the other robots for their local KB update. This processing is depicted in Figure 2. At the end of the goal accomplishment, the robot switches to the idle status to consider a new auctioning step. This information is sent to the other robots which is useful when performing the distributed Matrix-based auctioning algorithm.

Communication allows robot to exchange the information concerning the status of execution and also the level of interruptibility allowing at the receipt of a list of goals to consider only robot that could accomplish the goals according to their current status and to synchronize their local KB, and to construct their local matrix.

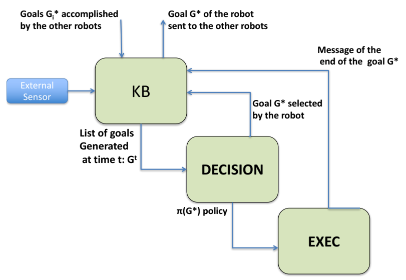

Goals G$_i$* accomplished by the other robots
Goal G* of the robot sent to the other robots
Message of the end of the goal G*
External Sensor
KB
Goal G* selected by the robot
List of goals Generated at time t: G^t
DECISION
π(G*) policy
EXEC

Figure 2: Communication between KB, decision and execution

It's possible that some messages contain values of some goals that the robot hasn't in its list. For these goals, the robot initializes the value of these goals in its value vector to $-V_{max}$.

3.3 Interruptibility and changing set of goals

In this section, we address the problem of changing goal sets. When the list of goals are communicated to the decision module, the robot can have different status. Indeed, the robot can be in a **idle status** or in a **active status**. When the robot is in a **idle status**, she is able to consider new goals and decides for the goals to accomplish. However, when the robot is in **active status**, the robot should consider the new goals only in the situations where the priority of one new goal is higher than the one under accomplishment.

All robots with priority lower than the priority of new goals, should consider them for a new auctioning. When the matrix-based auctioning algorithm of a robot has been performed and no goal is allocated, the robot pursues with the current policy, otherwise, the robot executes the policy of the new goal.

The priority of the robot is the one of the goal under processing. However, this priority can change during the execution since when the execution task is at its beginning stages, it's easier to cancel the execution and skip to another task rather than at its final execution stage. In our case, tasks are represented in an hierarchical structure called PRU+ as in [Iocchi et al., 2016]. Indeed, one benefit of the PRU+ structure is that goals could be accomplished partially and thus could be interrupted when the rest of the goal is not highly prior and in such case, we can enhance the PRU+ definition with the priority of levels.

4 Distributed Matrix-Based Auction Planning

4.1 General principle

The allocation of goals to the robots is performed by a distributed decision-theoretic market auction algorithm, extending the approach presented in [Spaan, Gonçalves, and Sequeira, 2010] to distributed auctioning and also using a distributed value function based on regret and opportunity cost to solve different potential conflicts on goals and robots. The proposed solution is illustrated in Figure 3 where each robot has a local auctioneer which receives the list of goals G from the local KB and sends this list to the decision-making module. This latter uses a library of task MDP models to solve for each goal in the list of goal its corresponding MDP by considering the current state of the robot. Thus the decision module of robot i solves $\{\ MDP_i(g, s_i^t) | \forall g \in G\ \}$. This allows the decision maker to compute the optimal value for each goal in the G $(v_i^{g_1}, v_i^{g_2}, \ldots, v_i^{g_k})$ for all $g_i \in G$. This vector of values is sent the local auctioneer that exchanges with the other local auctioneers. Once the local auctioneers collect all values and organize them in matrix 4. This matrix allows us to solve Equation (1). We consider in our approach the cases where this equation can have more than one solution meaning that more than one goal could be preferred by a robot and many robots can have one preferred goal leading to some conflicts situations which are not addressed in [Spaan, Gonçalves, and Sequeira, 2010].

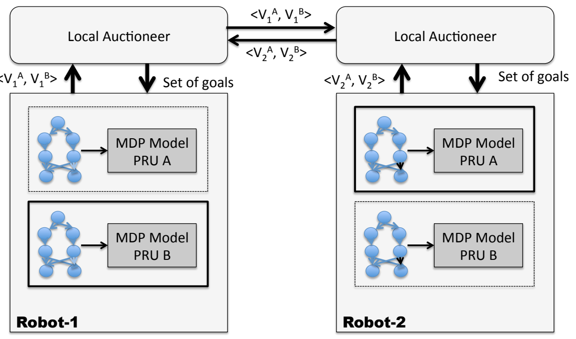


Figure 3: Each robot has an MDP for each goal represented by an acyclic PRU graph. The MDP of selected goal is active (solid box)

$$(\alpha, g^*) = argmax_{A_i, g_k} V_{A_i}(g_k) \qquad (1)$$

To address these issues, we introduce regret and opportunity cost functions. The regret function measures the loss in value for a robot to accomplish a goal rather than it's preferred and the opportunity cost of accomplishing a goal by a robot measures the loss in value of the other robots because of preventing them from this goal. More formally speaking, the *Regret* of not accomplishing a goal g^* is given by the following equation :

$$regret_j(g) = V_j^{\pi^*}(g) - \max_{g' \neq g} V_j^{\pi^*}(g')$$

And the opportunity cost is defined by :

$$OC_R(g) = \max_{R' \neq R} \max_{g' \neq g} V_{R'}^{*;g'} - V_{R'}^{*;g}$$

Let S_g be the set of robots α optimizing the value of accomplishing the goal g (solutions of Equation 1), the best robot to which we allocate the goal g is the one minimizing the regret (Equation 2). If we havemany solutions, we can proceed in the same way with the other goals and so on.

Let G_r be the set of the preferred goals of robot r. The best goal selected by robot r is the one minimizing the maximum opportunity cost (Equation 3).

4.2 Market-based auction algorithm for goal allocation using distributed value functions

The distributed market-based auction algorithm, illustrated in Figure 3, consists of two steps : Matrix construction step and Market-based auction step that we described in the following:

- **Matrix construction:** each robot i maintains a matrix M_i representing the value of the optimal policy of each robot to accomplish a goal. The matrix is constructed as follows:
  1. Each robot i computes the optimal value V_i^{*,g_l} to accomplish goal g_l. Value vector $(V_i^{*,g_1}, V_i^{*,g_2}, \ldots, V_i^{*,g_k})$ represents the values of robot i optimal policies accomplishing goals in the list. This vector represents the line i of the matrix.
  2. Each robot constructs locally this vector, communicates it to its local auctioneer and this latter sends its value vector to the others, allowing them to complete their matrix
  3. Each robot i (local auctioneer) has thus a matrix 4.
- **Distributed Matrix-based auctioning:** Each robot proceeds as depicted in 4 to the following steps:
  1. Fo each each line L of the matrix, compute $\max_j V_l^{*,g_j}$ corresponding to the best goal to accomplish for the robot L (its bid).
  2. If there is only one goal g_L^* maximizing the value of the robot L, this means that there is only a unique preferred goal for this robot. However, we should check that this robot is the preferred one for this goal. To this end, we check in the column g_L^* that there is no value $V_i^{*,g_L^*} \geq V_L^{*,g_L^*}$. In such case, the goal g_L^* is assigned to the robot L and ligne L and column g_L^* are removed from the matrix and we repeat the same process for the matrix until all goals have been assigned or all robots have an assigned goal.
  3. If there is more than one maximum value existing in columns or lines, we proceed as follows :
     (a) **Processing columns:** this means that there is a conflict between robots R on the same goals g. We assign the goal to the robot having the minimum regret ;

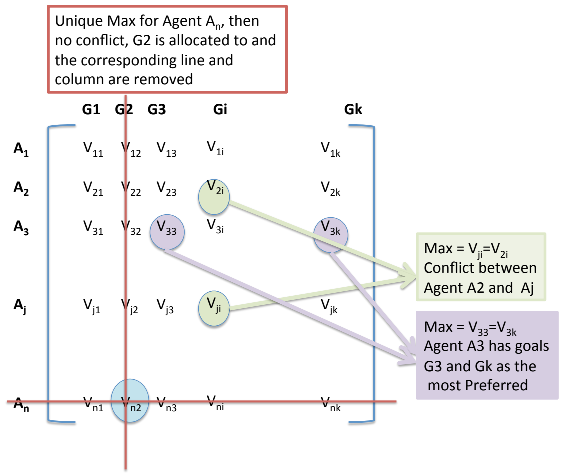
$$\text{argmin}_{robot \in R} \max_{g' \neq g} V_{robot}^{*,g'} - V_{robot}^{*,g} \qquad (2)$$

     Then we remove the adequate column and line.

     (b) **Processing lines:** this means that there are more than one goal G preferred by a robot R. In tis situation, we assign the goal with the minimum opportunity cost ;

$$\text{argmin}_{g \in G} OC_R(g) \qquad (3)$$

     Then we remove the adequate column and line.



Unique Max for Agent A$_n$, then no conflict, G2 is allocated to and the corresponding line and column are removed
G1 G2 G3 Gi Gk
A$_1$ V$_{11}$ V$_{12}$ V$_{13}$ V$_{1i}$ V$_{1k}$
A$_2$ V$_{21}$ V$_{22}$ V$_{23}$ V$_{2i}$ V$_{2k}$
A$_3$ V$_{31}$ V$_{32}$ V$_{33}$ V$_{3i}$ V$_{3k}$
A$_j$ V$_{j1}$ V$_{j2}$ V$_{j3}$ V$_{ji}$ V$_{jk}$
A$_n$ V$_{n1}$ V$_{n2}$ V$_{n3}$ V$_{ni}$ V$_{nk}$
Max = V$_{ji}$=V$_{2i}$ Conflict between Agent A2 and Aj
Max = V$_{33}$=V$_{3k}$ Agent A3 has goals G3 and Gk as the most Preferred

Figure 4: Matrix allocation processing

Proposition 1 *Matrix-based auctioning algorithm is based on solving $|G|$ MDPs with the same state space but with different goal states. The complexity of Matrix-based auctioning algorithm is $O(|G| \cdot |S|^2 \cdot |A|)$*

Proof Matrix-based auctioning algorithm solves one MDP for each goal of the set of goals $|G|$ with the same state space but with different goal states. The complexity of solving and MDP with value iteration algorithme is $O(|S|^2 \cdot |A|)$.

Proposition 2 *The cost of communication in Distributed Matrix-based Auctioning algorithm is in O(n^2) while in auctioning POMDP is in $O(3 \cdot n)$.*

Proof In Auctioning POMDP, each robot communicates with the central auctioning module by receiving the set of goals and sending its bid. The auctioning module sends n message to announce the goals, n bid messages are received from robots and sends n messages to robots for the auctioning result. In Distributed Matrix-based Auctioning algorithm, robots exchanges the goals and their bids as values of their optimal policies to accomplish these goals (individually). These message exchanges consist of $n-1$ messages sent by each robot (n robots). Thus the overall number of exchanged messages is $n \cdot (n-1)$. Thus the complexity is O(n^2).

4.3 The overall algorithm

1. The status of the Robot i is *Idle* :
   (a) Proceed to the message processing and produce the list of goals and sends local messages to the other robots;
   (b) Compute for each goal g in the list the value of the optimal policy V_g^* and put $Matrix(i,g) = V_g^*$;
   (c) Send the vector value to the decision module;
   (d) The decision module selects the best goal $g^* =$ Market_Based auction(i,g) (section 4.2) and sends the message to the KB;
   (e) The robot constructs the policy $\pi(g^*)$;

(f) The robot sends to the EXEC module and it changes its status to *active*;

2. The status of the robot i is *active*:
   (a) Transforms the policy in a PNP (Petri-Net Plan) as described in [Iocchi et al., 2016];
   (b) Execute the PNP and sends at each step of the PNP, the level of interruptibility;
   (c) At the end of the execution, sends the message to the local KB and changes the status to *idle*;

5 Dis-coordination minimization with lack of communication

When communication is not available two aspects have to be considered: (1) external sensor cannot communicate with robots and only onboard sensor of robots can be used leading to the lack of global perception and (2) robots cannot communicate and thus cannot exchange values of goals for auctioning. In order to overcome these limitations, we propose a simple policy estimation algorithm allowing each robot i to estimate the policy of each other robot $\pi_{j\neq i}$ without communication. We consider that the policy π_k^τ followed by robot k is the policy computed from the $MDP_k(s_k^\tau, G^\tau)$ where s_k^τ is the initial state of robot k and G^τ is the set of goals at time τ by selecting the policy maximizing the expected value of accomplishing one goal. In order to estimate the policy $\pi_{j\neq i}^t$ followed by each robot $j \neq i$ at time t, we need to estimate the policies followed by the robots during the interval $[t', t]$. However, during this interval one robot k can accomplish more than one goal and thus has followed a sequence of policies, noted $\mathcal{S}_k^\pi$ starting with $\pi_k^{t'}$ which is the policy of the $MDP_k(s_k^{t'}, G^{t'})$ where $(s_k^{t'}, G^{t'})$ are given by the local KB. However, to compute the next policies we need to derive the new MDP by deriving the new start state and the new set of goals. To do so, we approximate the new start state of k by s_k^τ as the most likely state can be reached by the policy $\pi_k^{t'}$ and that the new set of goal $G^\tau = G^{t'} - goal_{\pi_k^{t'}}$ where $goal_{\pi_k^{t'}}$ is the goal accomplished by $\pi_k^{t'}$ and finishing at time τ. More formally :

$$s_k^\tau = \max_{s_k'} P(s_k'|\pi_k^{t'}, s_k^{t'})$$

When s_k^τ and G^τ are derived, we can compute the next policy π_k^τ of $\mathcal{S}_k^\pi$ from the $MDP_k(s_k^\tau, G^\tau)$. We repeat the same processing until $\tau \geq t$. The last policy of $\mathcal{S}_k^\pi$, noted π_k^τ allows to estimate the s_k^τ and thus we compute the policy π_k^t of the $MDP_k(s_k^\tau, G^t)$ with the current set of goals. This principle is repeated for all robots and thus we get an approximate joint policy where each robot is assumed accomplishing a goal in G^t. Robot i compute a policy π_i^t of the $MDP_i(s_i^t, G^t - \bigcup_{k\neq i} goal_{\pi_k^t})$.

Proposition 3 *The complexity of approximating the policy of the other robots during an interval of time is $O(|S|^3 \cdot |A| \cdot |G|^2)$ where S and A are respectively the state and action spaces and G the set of goals at the lost of communication.*

Proof Solving an MDP with a goal and with an initial state using value iterationis O($|A| \cdot |S|^2$) [Puterman, 1994]. When we extend this to a set of goals where we should select the best goal, the complexity becomes $|G| \cdot |S| \cdot |A|^2$. To consider this problem for any initial state is in $|G| \cdot |S|^3 \cdot |A|$ because we repeat the same problem for each state and extending this problem for an interval of time is at most the resolution of all goals and thus the complexity becomes $|G|^2 \cdot |S|^3 \cdot |A|$. When we solve this problem for any state space and any set of goals, then for any robot we need just to know its initial state and which set of goals. Thus the complexity remains O($|G|^2 \cdot |S|^3 \cdot |A|$).

6 Empirical evaluation

We developed experiments where we consider our application of the shopping mall with 3 robots and a dozen of goals to accomplish. We compare our algorithm with the baseline algorithm of solving a DEC-MDP with a central planner as presented in [Hanna and Mouaddib, 2002] and an auctioning MDP approach using the principle presented in [Spaan, Gonçalves, and Sequeira, 2010]. We used different situations where goals are not located at the same place, for example for goals "guiding to shop" we consider different shops.

- We compare the computation time of each method: DEC-MDP, auctioning MDP and Decentralized Market-based Auctioning algorithm.
- We compare the values of the three approaches
- We compare the performance of these approaches when the set of goals changes by adding two more goals to the current list.
- We also compare the performance of these approaches when the communication is broken.

6.1 First results on performance comparison

The first results on computation time for three robots show that the computation time of DEC-MDP is higher than the others as expected which is explained by the complexity of DEC-MDP known to be NEXT-hard while our approach (MBAA) and Auctioning MDP have comparable computation time and MBAA is little bit higher due to the cost of communication.

The expected value of AMDP and MBA are the sum of the policy values of different goals accomplished by the three robots while for the DEC-MDP we get a value of a joint policy for the three robots. We observed that AMDP and MBAA are not far from a centralized planning approach representing more than 88% which is a satisfying performance. However, our approach outperforms the AMDP because it solves better the conflicts. We note that when conflicts doesn't occur, AMDP and MBAA obtain the same value.

To get more significant results more experiments are needed with more scenarii. Actually, we are continuing our evaluation.

6.2 Robustness to the changes of the goal set

The first experiment consists in starting with a list of 6 goals and during the execution of the policies we add two new goals. For this experiment, DECMDP approach considers the goals only when she finishes the accomplishment of the current goal and then consider the two new goals. The DEC-MDP plans from scratch for 7 goals (5 remaining goals plus the two new goals). In addition to that, the system should wait during the execution time which depends on the goal under execution (and the location of the shop). The AMDP finishes the execution of the current goal and considers the new one as DECMDP approach. This situation occurs in MBAA only when the priorities of the goals are lower than the one under execution otherwise, the executed is interrupted and the more prior goals are considered. A situation we observe during the experiment is when AMDP, DECMDP and MPAA are executing an advertisement goal and a new assistance a person goal arrive, the person should wait more than 3mn, 2mn and 30s respectively with DECMDP, AMDP and MPAA before being considered. These durations doesn't consider the execution time needed to meet the person (moving to the person).

6.3 Robustness to communication

DECMDP with no communication with the central planner they cannot work since they never receive their policy. AMDP needs communication with the central auctioneer and thus no solution is possible. However, MBAA can work with degraded more where only local planning and estimation of the other robots situations are performed. We use an experiment with 3 robots and 12 goals with two classes of configurations: Configuration A considers that goals scattered in the mall and configuration B considers goals in a narrow space. For configuration A, the approximation works well and only one conflict is observed (one dis-coordination) at the accomplishment of the two last goals where two robots head the same destination. However, in configuration B, we observe 4 conflicts (30%) where robots select the same goals. More deeper experiments in different situations are needed and let for future work.

6.4 Implementation on real-robots

The experiment on real robots as depicted in Figure 5 have been developed in assistance mission of visitors of our Lab where people can interact with the robot to ask for the office of a professor or administrative staff and then the robot guide him to the requested destination. When the robot accomplishes a goal, she comes back to the welcoming point to serve a new visitor depending on its location and the location of the visitor. Figure 5 shows that the same robot doesn't serve at the same point but according to their locations, the robot selects the appropriate welcoming point showing a very satisfying behavior. A video of this scenario is available at `https://youtu.be/iFC6-sCL3XI`.

7 Conclusion and discussion

We presented a practical approach allowing a fleet of robots to reason on local information and plan their activities in a coordinated way while activities and information are distributed in the environment. We develop a distributed Market-based auctioning method robust to the changes of the goal set and to the communication unavailability showing a very satisfying behavior. This method combines decision-theoretic planning

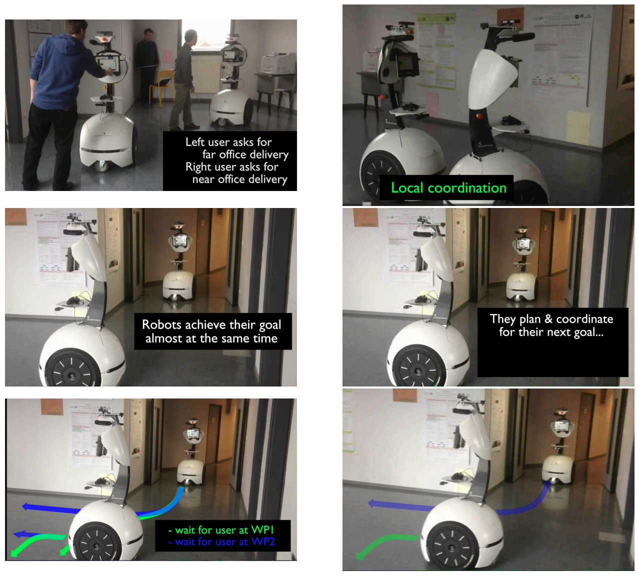


Figure 5: Figure showing different steps of robots serving visitors

by computing MDP policy for goals and bidding with values of these policies to coordinate their activities. Our method enriches the auctioning POMDP technique [Spaan, Gonçalves, and Sequeira, 2010] by considering distributed local auctioneers but also to use distributed value function based on regret and opportunity costs to solve some specific costs. Hoplites [Kalra, Ferguson, and Stentz, 2005] addresses similar problem but it is limited to path planning tasks in the opposite to ours. The dis-coordination minimization technique shows satisfying results and future work will be concerned by this aspect to deepen this problem and propose more efficient approach. Our approach has been successefuly used on real robots showing a very convincing behavior. We will develop more experiments to better evaluate the performance.

Aknowledgments:

This work has been developed in the COACHES project of the CHIST-ERA program supported by the French National Agency (ANR).

References

[Amato, Bernstein, and Zilberstein, 2007] Amato, C.; Bernstein, D. S.; and Zilberstein, S. 2007. Optimizing memory-bounded controllers for decentralized pomdps. In *UAI 2007, Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence, Vancouver, BC, Canada, July 19-22, 2007*, 1–8.

[Bernstein, Zilberstein, and Immerman, 2000] Bernstein, D. S.; Zilberstein, S.; and Immerman, N. 2000. The complexity of decentralized control of markov decision processes. In *UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, California, USA, June 30 - July 3, 2000*, 32–37.

[Dibangoye et al., 2016] Dibangoye, J. S.; Amato, C.; Buffet, O.; and Charpillet, F. 2016. Optimally solving dec-pomdps as continuous-state mdps. *J. Artif. Intell. Res.* 55:443–497.

[Ghallab, Nau, and Traverso, 2016] Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

[Hanna and Mouaddib, 2002] Hanna, H., and Mouaddib, A. 2002. Task selection problem under uncertainty as decision-making. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, 1303–1308.

[Iocchi et al., 2016] Iocchi, L.; Jeanpierre, L.; Lazaro, M. T.; and Mouaddib, A. 2016. A practical framework for robust decision-theoretic planning and execution for service robots. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, 486–494.

[Kalra, Ferguson, and Stentz, 2005] Kalra, N.; Ferguson, D.; and Stentz, A. 2005. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005, April 18-22, 2005, Barcelona, Spain*, 1170–1177.

[Matignon, Jeanpierre, and Mouaddib, 2012] Matignon, L.; Jeanpierre, L.; and Mouaddib, A. 2012. Coordinated multi-robot exploration under communication constraints using decentralized markov decision processes. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.*

[Nair et al., 2003] Nair, R.; Tambe, M.; Yokoo, M.; Pynadath, D. V.; and Marsella, S. 2003. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, 705–711.

[Nair et al., 2005] Nair, R.; Varakantham, P.; Tambe, M.; and Yokoo, M. 2005. Networked distributed pomdps: A synergy of distributed constraint optimization and pomdps. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, 1758–1760.

[Puterman, 1994] Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1st edition.

[Seuken and Zilberstein, 2007] Seuken, S., and Zilberstein, S. 2007. Memory-bounded dynamic programming for dec-pomdps. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, 2009–2015.

[Sonu and Doshi, 2015] Sonu, E., and Doshi, P. 2015. Scalable solutions of interactive pomdps using generalized and bounded policy iteration. *Autonomous Agents and Multi-Agent Systems* 29(3):455–494.

[Spaan, Gonçalves, and Sequeira, 2010] Spaan, M. T. J.; Gonçalves, N.; and Sequeira, J. 2010. Multirobot coordination by auctioning pomdps. In *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, 1446–1451.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Learning and Generalisation of Primitives Skills Towards Robust Dual-arm Manipulation

Èric Pairet and **Paola Ardón** and **Frank Broz** and **Michael Mistry** and **Yvan Petillot**
Edinburgh Centre for Robotics
University of Edinburgh and Heriot-Watt University, UK
{eric.pairet, paola.ardon}@ed.ac.uk, f.broz@hw.ac.uk, mmistry@inf.ed.ac.uk, y.r.petillot@hw.ac.uk

Abstract

Robots are becoming a vital ingredient in society. Some of their daily tasks require dual-arm manipulation skills in the rapidly changing, dynamic and unpredictable real-world environments where they have to operate. Given the expertise of humans in conducting these activities, it is natural to study humans' motions to use the resulting knowledge in robotic control. With this in mind, this work leverages human knowledge to formulate a more general, real-time, and less task-specific framework for dual-arm manipulation. The proposed framework is evaluated on the iCub humanoid robot and several synthetic experiments, by conducting a dual-arm pick-and-place task of a parcel in the presence of unexpected obstacles. Results suggest the suitability of the method towards robust and generalisable dual-arm manipulation.

INTRODUCTION

The last decades have witnessed a drastic increase in the use of robots in industry, professional and domestic environments. Among the countless competences that robots have acquired, some of the most outstanding are automating repetitive and exhausting tasks in manufacturing plants, working in hazardous scenarios unreachable to humans, assisting doctors in challenging surgical operations, and taking the responsibility for household chores. A common issue in all these applications is the need of manipulating large objects and ensembling multi-component elements without external assistance. On top of that, current manipulators lack human-like generalisation capabilities to confront the highly dynamic and changing environments. Thus, endowing robots with human-like dual-arm manipulation capabilities is essential to extend their competences and autonomy.

Traditional approaches for governing these dual-arm systems depend upon a great understanding of the model underlying the system's behaviour (Smith et al. 2012). Even though deriving an accurate model is possible for some complex systems, approximations are commonly used in order to make the calculations computationally tractable, despite the trade-off of the model's uncertainty (Pairet et al. 2018). Furthermore, some of these methods lack scalability and generalisation capabilities along and across tasks. In other words, they require an expert programmer to hand-define all possible scenarios, movements, tasks, and extensive manual tuning of the system's control architecture (Argall et al. 2009).

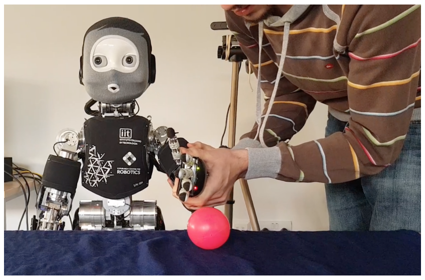


Figure 1: iCub humanoid being taught how to avoid an obstacle (red sphere). Within the proposed framework, this primitive skill provides robustness to novel scenarios.

The growth of artificial intelligence (AI) has popularised more natural techniques for robot learning, reducing the laborious task of coding every possible scenario and thus, increasing modularity and flexibility on the systems. An example of this is imitation learning or learning by demonstration (LbD). This method allows non-robotics-experts to interact, teach and modify the robot's behaviours (Nicolescu and Mataric 2003), and, consequently, to obtain more human-like behaviours with enhanced acceptability and compatibility to the human workspaces (Ajoudani et al. 2017). Given the possibility to learn from humans' expertise and dexterity in using both arms for manipulation purposes, it is natural to exploit LbD to use human motions in robotic control.

Teaching a robot from human demonstrations can be challenging. The different anatomical characteristics between the teacher and the learner produces the correspondence problem, i.e. the issue of identifying a mapping between the teacher and the learner which allows transferring of information from one to the other (Dautenhahn and Nehaniv 2002). Moreover, complex motions involve a mixture of human intentions, which are difficult to accurately learn when following an all-at-once learning baseline (Bajcsy et al. 2018). On top of that, teaching a dual-arm system can suppose a high endeavour for non-robotics-experts (Akgun et al. 2012).

LbD offers some generalisation capabilities, such as changes in initial and goal configurations of a given demonstration (Billard et al. 2008). However, being limited to similar scenarios is not realistic to the rapidly changing, dynamic and unpredictable environments where robots have to operate. Extended robustness can be obtained by letting the system iteratively adapt and improve the learnt task to new scenarios (Guenter et al. 2007). This leads to the well-known exploration-exploitation dilemma and comes at the cost of needing to fail in order to learn and consequently, at the risk of causing harm to the robot during the self-learning process.

This paper presents a framework that seeks to jointly overcome the aforementioned issues, namely (i) the complex and ambiguous teaching procedures and (ii) the limited generalisation capabilities. Aiming to provide a dual-arm system with a more general and less task-specific method for real-time and robust manipulation in challenging even unfamiliar environments, the proposed framework (i) leverages human knowledge to learn and create a library of primitive skills and (ii) endows dual-arm systems with human-like manipulation capabilities by combining the primitive behaviours.

The main contribution of this work is the formulation of a framework which learns individual primitive skills from human demonstrations and exploits them for robust dual-arm manipulation purposes. Such a framework extends the capabilities of the method in (Pastor et al. 2009) to handle the requirements of dual-arm systems. This leads to a framework which reuses its knowledge to generalise according to the environment awareness, differently from the proposals in (Zöllner, Asfour, and Dillmann 2004; Topp 2017). The potential of this method has been demonstrated in a simulated environment on the iCub humanoid (see Figure 1). The experimental results suggest the suitability of the framework to address the aforementioned challenges.

SYSTEM DESCRIPTION AND PROBLEM FORMULATION

The aim of this paper is an end-to-end learning-based framework that allows real-time autonomous dual-arm manipulation in unfamiliar environments. Such a framework needs to be able to adapt its plan to achieve a task according to the surrounding environment, while ensuring some synchronisation between both end-effectors. Moreover, it needs to be easily programmable, making a dual-arm platform customizable and accessible even to non-robotics-experts. Bearing these problem requirements in mind, this section firstly describes the typology and diversity of possible actions in a dual-arm system. It then analyses the challenges that arise when learning actions from human demonstrations. Finally, this section puts the previous pieces together to formulate the modelisation of the dual-arm system and its grasping.

Dual-arm Primitive Skills Taxonomy

Dual-arm manipulators are extremely sophisticated systems, and consequently, their control actions to achieve a specific performance. This work contemplates that any complex behaviour is composed of a vast repertoire of actions or primitive skills (Montesano et al. 2008). In the context of manipulation via a dual-arm system, a possible classification of any primitive skill falls into these two groups:

- Absolute skills $\mathcal{S}_a$: imply a change of configuration of the manipulated object in the Cartesian space. Example: move, place and/or turn an object in a particular manner.
- Relative skills $\mathcal{S}_r$: exert an action on the manipulated object in the Object space. Example: opening of a bottle's screw cap, or hold a parcel by means of force contact.

Each type of primitive skill uniquely produces movement in its own space. In other words, the absolute and relative skills lay on orthogonal spaces. It is natural to expect from a dual-arm system to simultaneously carry out, at least, one absolute and one relative skill to successfully accomplish a task. Let us analyse the task of moving a bottle to a certain position while opening its screw cap. Both end-effectors synchronously move to reach a desired configuration (absolute skill). At the same time, the left end-effector is constrained to hold the bottle upright (relative skill), while the right end-effector unscrews the cap (relative skill).

Learning for a Dual-arm Manipulator

Learning by demonstration (LbD) provides a large set of recording techniques and mathematical supports for encoding a demonstrated skill. However, learning a particular task from human demonstrations raises some challenges, namely (i) clearly understanding the intentions of a demonstration and (ii) establishing a teacher-learner communication channel. Both issues can drastically affect the learning outcome if they are not well adressed (Argall et al. 2009).

The demonstration clarity issue is tackled by leveraging the belief of a vast repertoire of primitive skills being the basis of any complex behaviour. With this in mind, this work avoids demonstrating a task itself but, instead, teaches the robot the involved primitive skills. This task factorisation provides similar benefits as the work in (Bajcsy et al. 2018): it allows the user to teach one feature of the task at a time, and, if required, to correct them individually.

Factorising a complex behaviour into primitive actions reduces the number of degree of freedom (DoF) to focus on during demonstration time. As an example, the desired position and orientation of a task can be encoded in separate primitive skills and thus, demonstrated one-at-a-time. This fact becomes handy to ease the complex process of teaching a dual-arm system (Akgun et al. 2012). This work employs kinesthetic guiding to establish a teacher-learner communication channel which does not suffer from the correspondence problem (Argall et al. 2009).

Dual-arm System Modelisation

Given the variety of primitive skills that a dual-arm system can execute, this work seeks to model the robotic platform in a generalisable yet modular fashion, which accounts for both absolute and relative skills. To this aim, let us consider the closed kinematic chain depicted in Figure 2. Each arm i, where $i = \{L,\ R\}$, interacts with the same object $\mathcal{O}$ in the workspace $\mathcal{W} \in \mathbb{R}^N$, where N is the dimensionality of the considered Cartesian subspace. In this context, the absolute skill explains the movement of the object $\mathcal{O}$ in the workspace $\mathcal{W}$, while the relative skill describes the actions of each end-effector i with respect to the object's reference frame $\{\mathcal{O}\}$. Note that $\{\mathcal{O}\}$ is the centre of the closed-chain dual-arm system. Thus, the remaining of the paper uses $\{\mathcal{O}\}$ as object's and system's frame indistinguishably.

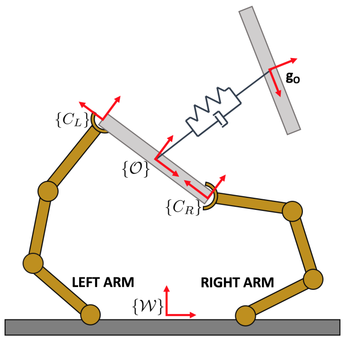


Figure 2: Dual-arm manipulator modelled in the Cartesian space as a spring-damper closed-chain system.

Let the current state of the closed-chain dual-arm system be defined by the position, velocity and accelaration of its system's frame $\{\mathcal{O}\}$ in each DoF of the workspace $\mathcal{W}$, i.e. $(x_o,\ \dot{x}_o,\ \ddot{x}_o)_n\ \forall\ n \in [1,\ N]$. The dynamics of such a system are approximated by the ones of a spring-damper system acting between the objects's frame $\{\mathcal{O}\}$ and its goal configuration $\mathbf{g}_o$ (see Figure 2). This dynamical system genarates in each DoF a movement trajectory $\mathbf{x}_o$ with velocity $\dot{\mathbf{x}}_o$ and acceleration $\ddot{\mathbf{x}}_o$ profiles defined by:

$$\tau\ddot{\mathbf{x}}_o = \alpha(\beta(g_o - \mathbf{x}_o) - \dot{\mathbf{x}}_o), \tag{1}$$

where g_o is the model's attractor that the system will converge to with critically damped dynamics and null velocity when $\alpha > 0$, $\beta > 0$ and $\beta = \alpha/4$ (Ijspeert et al. 2013).

Given any initial system state, the dynamical system in (1) generates a linear displacement towards the goal configuration $\mathbf{g}_o$. Any other dynamical behaviour can be represented by an external force acting on the system's frame $\{\mathcal{O}\}$ as:

$$\tau\ddot{\mathbf{x}}_o = \alpha(\beta(g_o - \mathbf{x}_o) - \dot{\mathbf{x}}_o) + \mathbf{f}_o(\cdot), \tag{2}$$

where the coupling term $\mathbf{f}_o(\cdot)$ describes the profile of the external force affecting the natural dynamics of the system. In other words, $\mathbf{f}_o(\cdot)$ characterises the system's behaviour and thus, can be used to encode and retrieve a primitive skill.

Dual-arm Grasping Geometry

Any action referenced to the object's frame $\{\mathcal{O}\}$ can be projected to the end-effectors using the grasping geometry of the manipulated object. This allows computing the required end-effector control commands to achieve a particular absolute task. To this aim, the grasp matrix needs to be computed. The grasp matrix $\mathbf{G}_i$ of the end-effector i is a transformation map which establishes a velocity relation between the contact point C_i, and the systems reference frame $\{\mathcal{O}\}$. For a workspace $\mathcal{W}$ of $N = 6$, i.e. considering the linear and rotational information of the three-dimensional (3D) space, the grasping geometry establishes the following relation:

$$\dot{\mathbf{x}}_{C_i} = \mathbf{G}_i^T\ \dot{\mathbf{x}}_o, \tag{3}$$

where

$$\mathbf{G}_i \in \mathbb{R}^{6\times6} = \begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{O}_{3\times3} \\ \mathbb{S}(\mathbf{r}_i) & \mathbf{I}_{3\times3} \end{bmatrix}, \tag{4}$$

where $\mathbf{I}_{3\times3}$ is the identity matrix, and $\mathbb{S}(\mathbf{r}_i) \in \mathbb{R}^{3\times3}$ is the skew-symmetric matrix performing the cross product:

$$\mathbb{S}(\mathbf{r}_i) = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}, \tag{5}$$

where $\mathbf{r}_i$ is the distance from the object's reference frame $\{\mathcal{O}\}$ to the contact point C_i.

A global grasp map $\mathbf{G}$ for the dual-arm manipulator can be defined by horizontally concatenating the grasp matrix of each end-effector, i.e. $\mathbf{G} = [\mathbf{G}_L\ \mathbf{G}_R] \in \mathbb{R}^{6\times12}$ where $\mathbf{G}_L$ and $\mathbf{G}_R$ are the left and right arm grasp matrix, respectively.

FRAMEWORK FOR ROBUST DUAL-ARM MANIPULATION

In order to endow robots with real-time, robust and autonomous dual-arm manipulation, while letting non-robotics-experts to easily program and customise the system's behaviour, this work presents the learning-based framework depicted in Figure 3. Such a framework jointly addresses the aforementioned requirements with three sequential parts: (i) the learning module that learns a set of primitive skills from human demonstrations, (ii) the roll-out module that combines those primitive skills to plan a trajectory which makes the system succeed at a task, even in unfamiliar environments and (iii) the evaluation module that lets a human-in-the-loop supervise the robot's behaviour and reteach a specific skill, if required.

Given a learnt repertoire (library) of absolute and relative primitive skills, such basic motions need to be combined to confront any dual-arm task in any possible scenario. Each absolute task $\mathcal{S}_a$ is defined by its coupling term $\mathbf{f}_o(\cdot)$, which leads to a desired triplet $(\ddot{\mathbf{x}}_o,\ \dot{\mathbf{x}}_o,\ \mathbf{x}_o)_n\ \forall\ n \in [1,\ N]$ after rolling-out (2). Similarly, each relative task $\mathcal{S}_r$ defines a desired triplet for each end-effector i $(\ddot{\mathbf{x}}_{C_i},\ \dot{\mathbf{x}}_{C_i},\ \mathbf{x}_{C_i})_n\ \forall\ n \in [1,\ N]$. This work considers weighting and merging the contribution of each primitive skill at the velocity level as:

$$\begin{bmatrix} \dot{\mathbf{x}}_L \\ \dot{\mathbf{x}}_R \end{bmatrix} = \mathbf{G}^T \sum_{j=1}^{J} w_j\ \dot{\mathbf{x}}_{o_j} + \sum_{k=1}^{K} w_k \begin{bmatrix} \dot{\mathbf{x}}_{C_{L,k}} \\ \dot{\mathbf{x}}_{C_{R,k}} \end{bmatrix}, \tag{6}$$

where $\dot{\mathbf{x}}_L$ and $\dot{\mathbf{x}}_R$ respectively are the velocity commands for the left and right end-effector which satisfy the set of activated primitive skills, $\dot{\mathbf{x}}_{o_j}$ is the velocity of the $j \in [1,\ J]$ absolute primitive skills available in the library, and $\dot{\mathbf{x}}_{C_i,k}$ is the velocity of the $k \in [1,\ K]$ relative primitive skills available in the library. Primitive skill selection according to a desired task and environment is conducted with the weights w_j and w_k. Works such as the one in (Ardón et al. 2018) propose addressing this problem according to the object's affordances and environment analysis.

The generality of the proposed framework is narrowed down to provide an application case. This work exploits such a framework to endow a dual-arm system with enhanced autonomy for the dual-arm task of pick-and-place of a parcel, even in the presence of unexpected obstacles. Figure 4 depicts the main idea: parcels (brown prisms) are meant to be

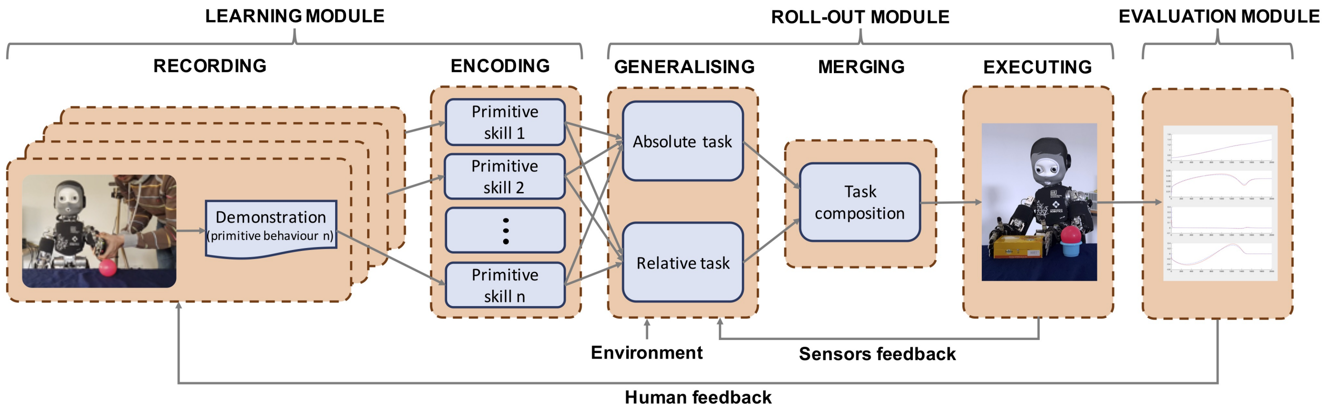


Figure 3: Scheme of the three stages involved in the proposal. Learning: a human demonstrator teaches some primitives behaviours to a system. Rolling-out: the robot exploits (generalises and combines accordingly to the environment awareness) the acquired knowledge. Evaluation: an evaluator inspects the system's performance and decides whether reteaching is necessary.

moved from one side to another, adjusting the behaviour of the dual-arm whether there is an obstacle or not (grey prism). Not requiring complex grasping capabilities is the main reason for choosing this application case. However, it is extremely challenging in the synchronisation aspect: manipulators have to always maintain a certain amount of contact forces with the carried parcel as any variation would result in releasing or exposing the handled object to stress. To this aim, the library of primitive skills is loaded with: underlying dynamics of a pick-and-place task, obstacle avoidance and grasp maintenance behaviours. Note that the former two skills are absolute, while the latter is relative.

Skill Dynamics

The non-linear dynamical behaviour of any task can be represented using dynamic movement primitives (DMPs). This mathematical encoding support has proven to be a versatile tool for modelling and learning complex motions, since: (a) any movement can be efficiently learned and generated, (b) a unique demonstration is already generalisable, (c) convergence to the goal is guaranteed, and (d) their representation is translation and time-invariant (Pastor et al. 2009; Ijspeert et al. 2013). Some of these DMP-inherent generalisation capabilities are depicted in Figure 5.

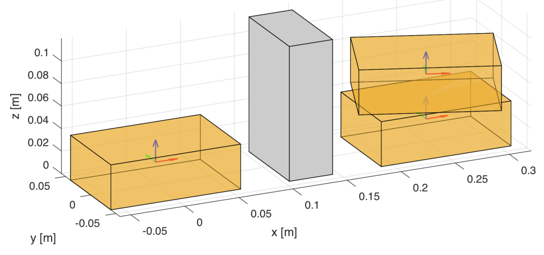


Figure 4: Dual-arm pick-and-place of a parcel (brown prism) in the presence of obstacles (grey prism).

The system modelisation in (2) can integrate a DMP as the coupling term $\mathbf{f}_o(\cdot)$. This means that the perturbation-less dynamics of the spring-damper system are modified according to the DMP coupled in each DoF. If $\mathcal{W} \in \mathbb{R}^3$, three position-encoding DMPs would describe the desired position of the manipulated object. Instead, if $\mathcal{W} \in \mathbb{R}^6$, four additional quaternion-based DMPs would be required to also encode the object's desired orientation (Ude et al. 2014).

Formally, a position-encoding DMP is a weighted linear combination of non-linear radial basis functions (RBFs) (Pastor et al. 2009; Ijspeert et al. 2013). The value of such non-linear function $\mathbf{f}_o(\cdot)$ when evaluated at a specific entry $k \in \mathbf{k}$ is defined as:

$$f(k) = \frac{\sum_{i=1}^{N} w_i \Psi_i(k)}{\sum_{i=1}^{N} \Psi_i(k)} \, k, \tag{7}$$

$$\Psi_i(k) = \exp\left(-h_i(k - c_i)^2\right), \tag{8}$$

where c_i and $h_i > 0$ are the centres and widths, respectively, of the $i \in [1, \ N]$ RBFs distributed along the trajectory. Each RBF is weighted by w_i. The phase variable $\mathbf{k}$ is utilised to avoid direct dependency of $\mathbf{f}_o(\cdot) \sim \mathbf{f}(\mathbf{k})$ on time. The dynamics of $\mathbf{k}$ are defined as:

$$\tau \dot{\mathbf{k}} = -\alpha_k \mathbf{k}, \tag{9}$$

where the initial value of the canonical system $\mathbf{k}(0) = 1$ and α_k is a positive constant.

The learning of the DMPs relies on adjusting the set of RBF, i.e. the weight vector $\mathbf{w}$, composed of all weights w_i. To this aim, least mean squares (LMS) is used to compute the weight vector $\mathbf{w}$ which makes the system (2) adjust to a recorded skill propioception information $\{\ddot{\mathbf{x}}, \ \dot{\mathbf{x}}, \ \mathbf{x}\}$.

Obstacle Avoidance

An analytical description of how humans steer around an obstacle was first presented in (Fajen and Warren 2003). Later on, such biologically-inspired formulation was used in (Hoffmann et al. 2009) for single-arm manipulation purposes. Let $\mathbf{x}$, $\dot{\mathbf{x}}$, and θ be respectively the system's position,

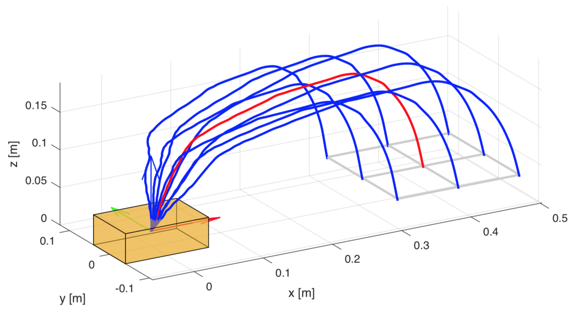


Figure 5: DMP generalisation capabilities. Given a demonstration (red trajectory), rolling-out (2) with the DMP coupling term $\mathbf{f}_o(\cdot)$ defined in (7) let the system generalise to new goal configurations (blue trajectories).

velocity and orientation referenced to the workspace reference frame $\{\mathcal{W}\}$. In order to avoid an obstacle, the system in (2) needs to change its acceleration accordingly to:

$$\mathbf{f}(\mathbf{x},\ \dot{\mathbf{x}}) = \mathbf{R}\,\dot{\mathbf{x}}\,\dot{\theta}, \tag{10}$$

where $\mathbf{R}$ is a $\pi/2$ rotation matrix with respect to the vector $\mathbf{r} = (\mathbf{x}_{obstacle} - \mathbf{x}) \times \dot{\mathbf{x}}$, and $\dot{\theta}$ is the desired turning velocity:

$$\dot{\theta} = \gamma\,\theta \exp(-\beta\,|\theta|), \tag{11}$$

where γ and β are tuning constants. Their effect can be best understood in Figure 6: γ sets the abruptness of the obstacle avoidance behaviour, and β determines its sensitivity.

Within the framework, the parameters of the obstacle avoidance behaviour are leant from human demonstrations, thus involving less parameter tuning. This is achieved using LMS after log-linearising (11) and arranging it as:

$$\log \dot{\boldsymbol{\theta}} = [\log\gamma \quad 1 \quad \beta] \begin{bmatrix} \mathbf{1} \\ \log \boldsymbol{\theta} \\ -|\boldsymbol{\theta}| \end{bmatrix}, \tag{12}$$

where the training data $\dot{\boldsymbol{\theta}}$ and $\boldsymbol{\theta}$ contain the periodically sampled value of $\dot{\theta}$ and θ experienced during the obstacle avoidance demonstration. The change in steering angle $\dot{\theta}$ is retrieved from (10), where $\mathbf{f}(\mathbf{x},\ \dot{\mathbf{x}}) = \mathbf{f}(\mathbf{x})_{obs} - \mathbf{f}(\mathbf{x})$, i.e. the difference on the dynamics between a perturbationless task $\mathbf{f}(\mathbf{x})$ and one with obstacles $\mathbf{f}(\mathbf{x})_{obs}$ is only motivated by the presence of an obstacle.

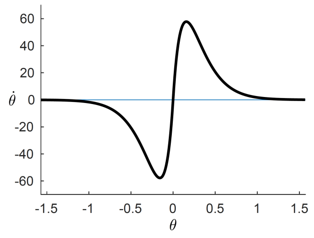


Figure 6: Change of steering angle $\dot{\theta}$ following the original formulation in (11) with $\gamma = 1000$ and $\beta = 20/\pi$.

Grasp Maintenance

Manipulation of a rigid object via a dual-arm system requires each end-effector to be in contact with the object. Moreover, when the interaction is by force contact (without grasping the object) it is also essential to apply the sufficient forces to ensure grasp maintenance, i.e. prevention of contact separation and unwanted contact sliding. The complexity of this task usually requires modelling the required coupling forces as $\mathbf{f}(\mathbf{x},\ \dot{\mathbf{x}},\ \ddot{\mathbf{x}})$. For applications with low-dynamical requirements, the previous dynamical function can be approximated to (Gams et al. 2014):

$$\dot{\mathbf{x}}_{C_i} = \mathbf{K}(\mathbf{F}_d - \mathbf{F}_r), \tag{13}$$

where $\mathbf{K}$ is an error multiplying constant which transforms errors in force contact to velocity commands, $\mathbf{F}_d$ is the desired coupling force and $\mathbf{F}_r$ is the current coupling force retrieved from the robot's sensors. Thus, the learning of this primitive skill resides on learning from demonstrations which $\mathbf{F}_d$ ensures grasp maintenance.

RESULTS AND EVALUATION

The work presented in this paper is a generic framework for any dual-arm manipulator. Experimental evaluation has been carried out on synthetic environments and a simulated iCub humanoid. This section firstly introduces the iCub robot and the execution of kinesthetic learning on this platform. It then describes the learning of the obstacle avoidance behaviour, and it analyses its integration in a synthetic pick-and-place task. Finally, this section depicts the potential of the proposed framework for being used on a humanoid robot.

Experimental Platform

iCub is an open source humanoid robot testbed for research into human cognition and artificial intelligence applications (Metta et al. 2008). The physical and software characteristics of this robot make it an ideal platform for the presented research. Among all this robot's features, some of the most relevant to this work are the two 7-DoF manipulators equipped with a torque sensor on the shoulder, tactile sensors in the fingertips and palm, and integrated stereo vision. iCub operates under the YARP middleware.

Kinesthetic teaching on the iCub humanoid is conducted by setting all joints in gravity compensation allowing the teacher to physically manoeuvre the robot through a desired skill. During the demonstrations, proprioception information is retrieved via YARP ports.

Obstacle Avoidance Behaviour

The primitive skill of obstacle avoidance has been taught to iCub with two different behaviours: reckless (see Figure 7(a)) and conservative (see Figure 7(b)). While the former steers around the obstacle (red sphere) closely, the latter keeps a larger distance to it. The recorded raw proprioception data of these two kinesthetic demonstrations is respectively portraited in Figure 7(c) and Figure 7(d). As it can be observed, the retrieved trajectories are noisy and not smooth.

To learn from these demonstrations, the data has been preprocessed in two steps: (i) filtering to remove outliers and

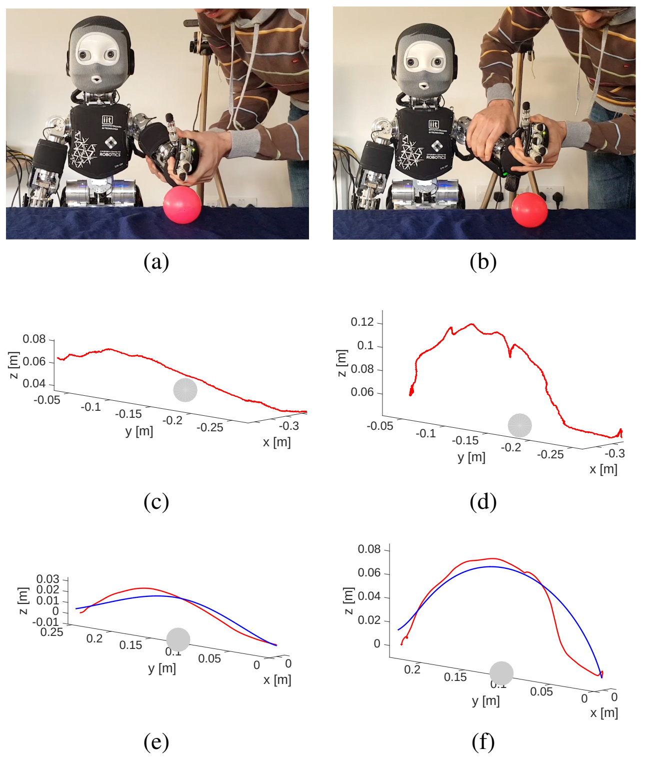


Figure 7: iCub humanoid robot (Metta et al. 2008) learning the primitive skill of obstacle avoidance with two different behaviours: reckless (first column) and convervative (second column). (a)-(b) Human demonstrations to avoid an obstacle (red sphere). (c)-(d) iCub's proprioception data. (e)-(f) Processed iCub's proprioception data (red trajectory) and learned obstacle avoidance behaviour (blue trajectory).

high-frequency noise, and (ii) projecting the filtered information to the two-dimensional (2D) space composed for the two principal components of the data. Figure 7(e) and Figure 7(f) show the preprocessed data (red trajectory), which has been used in (12) to learn the parameters defining the demonstrator's obstacle avoidance behaviour. The encoded reckless and conservative styles are respectively depicted in Figure 7(e) and Figure 7(f) (blue trajectory). Note that learning the parameters instead of the motion itself lets the robot generalise such behaviour under different conditions.

In overall terms, from Figure 7 it can be concluded that the obstacle avoidance encoding support and its learning process from human demonstrations is able to encapsulate the demonstrator style. The differences between the demonstrated skill and the learnt one are mainly attributed to the hypothesis that any steering around an obstacle follows the formulation in (10)-(11). Moreover, the noise in the proprioception data increases the variance in the learning. Alternatively, a high-precision tracking system such as the one used in (Rai et al. 2014) can be considered. Because the proposed approach extracts the parameters of an obstacle avoidance behaviour, the resulting knowledge would yet be independent of the demonstration frame.

Synthetic Pick-and-Place Task

The performance of the obstacle avoidance behaviour in a more realistic context has been validated using the pick-and-place setup depicted in Figure 8. Particularly, an initial pick-and-place demonstration is given to the system (red trajectory), consisting of moving the parcel from the left to the right without the presence of any obstacle (grey prism). The underlying dynamics defining this primitive skill have been encoded as a DMP. Due to the inherent generalisation capabilities of the DMPs, the system is already able to infer the pick-and-place dynamics from any different starting and goal positions (blue trajectory), but not able to generalise to the presence of obstacles. Only after coupling the previously learnt pick-and-place dynamics and obstacle avoidance behaviour together, the system is able to generalise in real-time to the presence of unexpected obstacles (black trajectory).

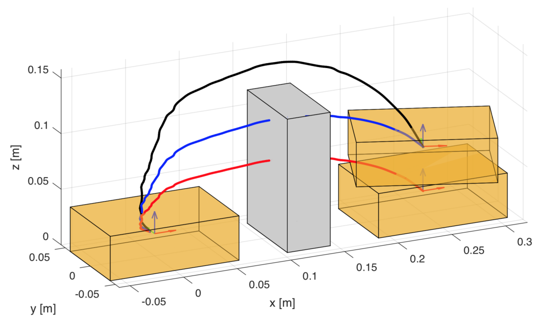


Figure 8: Dual-arm pick-and-place of a parcel (brown prism) in the presence of obstacles (grey prism). Demonstration (red trajectory), inference to a new position (blue trajectory), inference with obstacle avoidance (black trajectory).

Framework on Humanoid Robot

The applicability of the framework has been tested with a particular dual-arm task. The framework has been developed in YARP to deploy it on a simulated iCub humanoid. Due to the lack of realistic simulated force sensors and thus, lack of awareness of the exerted force on the carried object, the grasp maintenance skill primitive in (13) has been replaced according to the proposal in (Gams et al. 2014):

$$\dot{\mathbf{x}}_{C_i} = \mathbf{K}(\mathbf{D}_{d_i} - \mathbf{D}_{r_i}), \tag{14}$$

where $\mathbf{D}_{d_i}$ is the desired distance from the object's frame $\{O\}$ to the contact point C_i, being $i = \{L,\ R\}$, and $\mathbf{D}_{r_i}$ is the current distance. Due to the symmetry of the task, $\mathbf{D}_{d_L} = \mathbf{D}_{d_R}$. Thus, the learning of this primitive skill is reduced at setting $\mathbf{D}_{d_i}$ accordingly to the characteristics of the manipulated parcel and the grasping points.

After this arrangement forced by the simulated nature of the experimentation, the pick-and-raise activity was conducted (see Figure 9). Such a task consists of picking a parcel from the table and raising it with certain dynamics (red trajectory), while avoiding obstacles and ensuring grasp maintenance. iCub performed the described dual-arm task

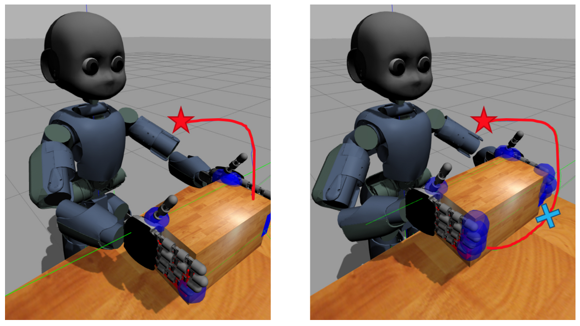

(a) (b)

Figure 9: iCub humanoid robot (Metta et al. 2008) picking a parcel and raising it with specific dynamics (red trajectory) to a goal configuration (red star). (a) Following the task dynamics previously learnt from a human demonstrator. (b) Modifying the task dynamics in real-time to avoid an obstacle (blue cross). Grasp maintenance is successfuly ensured in both cases by the corresponding primitive skill.

in two different contexts. First, with the absence of obstacles, where the robot can move the parcel with the designated dynamics (see Figure 9(a)). Second, with the presence of unexpected obstacles (blue cross), where the robot had to replan the trajectory to achieve the goal configuration (see Figure 9(b)). Despite the simplicity of the used primitive skill to ensure grasp maintenance, the trials were successful: both end-effectors were accurately synchronised so the handled parcel was neither released nor exposed to stress.

These results show that iCub has been able to perform the pick-and-place task even in the presence of an unexpected obstacle, after learning three primitive skills individually from a human demonstrator. This fact raises expectations about the degree of similarity that iCub's final behaviour might have with the demonstrator's behaviour under the same conditions. Analysing this similarity is of interest to the human-robot interaction (HRI) community, since it can contribute to enhancing the acceptability and compatibility of robots in human workspaces (Ajoudani et al. 2017). An alternative for conducting this study consists of recording some samples of the robotic and human approach to quantify their deviation with the Kullback-Leibler (KL) divergence statistic. The lower this indicator is, the higher the chances are that these two agents have similar behaviours. Such a study is left for future work.

FINAL REMARKS AND FUTURE WORK

This work has presented a novel framework which endows a dual-arm system with real-time, robust and less task-specific manipulation capabilities. Such a framework is twofold: (i) learns from human demonstrations to create a library of primitive skills, and (ii) combines such knowledge to confront challenging unfamiliar scenarios with human-like manipulation capabilities. Unlike the framework of motion primitives in (Pastor et al. 2009), the proposed approach handles primitive skills for dual-arm manipulation purposes while still being able to combine different primitives at the same time. This feature is what differentiates the current work from similar ones (Zöllner, Asfour, and Dillmann 2004; Topp 2017). The evaluation conducted on the iCub humanoid suggest the proposal's suitability for robust dual-arm manipulation, yet with some room for improvement.

The framework is not restricted to the presented experimental evaluation nor platform. Any system able to retrieve proprioception information can benefit from this work. Moreover, any primitive skill that might be required for dual-arm manipulation can be included in the framework's library. The application case reported in this manuscript exemplifies this fact by considering, among other primitive skills, an obstacle avoidance behaviour which steers around obstacles in real-time. The desired reactivity of this obstacle avoidance behaviour is learnt from human demonstrations.

Future work will significantly extend the library of primitive skills such that more tasks and scenarios involving challenging dual-arm manipulation tasks can be addressed within the framework. Action selection will be integrated to automatically select from the framework's library the necessary set of skills to address a specific task. Apart from the task itself, surrounding environment and characteristics and constraints of the object to manipulate might need to be considered. Finally, imminent efforts will focus on learning force-dependant primitive skills, such as the grasp maintenance one, on the real iCub humanoid robot, as well as evaluating the entire framework on such platform.

ACKNOWLEDGMENTS

This work has been partially supported by ORCA Hub EPSRC (EP/R026173/1) and consortium partners.

References

Ajoudani, A.; Zanchettin, A. M.; Ivaldi, S.; Albu-Schäffer, A.; Kosuge, K.; and Khatib, O. 2017. Progress and prospects of the human–robot collaboration. *Autonomous Robots* 1–19.

Akgun, B.; Cakmak, M.; Jiang, K.; and Thomaz, A. L. 2012. Keyframe-based learning from demonstration. *International Journal of Social Robotics* 4(4):343–355.

Ardón, P.; Pairet, È.; Ramamoorthy, S.; and Lohan, K. S. 2018. Towards robust grasps: Using the environment semantics for robotic object affordances. In *AAAI Fall Symposium. Reasoning and Learning in Real-World Systems for Long-Term Autonomy*. AAAI Press.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

Bajcsy, A.; Losey, D. P.; O'Malley, M. K.; and Dragan, A. D. 2018. Learning from physical human corrections, one feature at a time. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 141–149. ACM.

Billard, A.; Calinon, S.; Dillmann, R.; and Schaal, S. 2008. Robot programming by demonstration. In *Springer handbook of robotics*. Springer. 1371–1394.

Dautenhahn, K., and Nehaniv, C. L. 2002. The correspondence problem. In *Imitation in Animals and Artifacts, MIT Press*. MIT Press.

Fajen, B. R., and Warren, W. H. 2003. Behavioral dynamics of steering, obstable avoidance, and route selection. *Journal of Experimental Psychology: Human Perception and Performance* 29(2):343.

Gams, A.; Nemec, B.; Ijspeert, A. J.; and Ude, A. 2014. Coupling movement primitives: Interaction with the environment and bimanual tasks. *IEEE Transactions on Robotics* 30(4):816–830.

Guenter, F.; Hersch, M.; Calinon, S.; and Billard, A. 2007. Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics* 21(13):1521–1544.

Hoffmann, H.; Pastor, P.; Park, D.-H.; and Schaal, S. 2009. Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 2587–2592. IEEE.

Ijspeert, A. J.; Nakanishi, J.; Hoffmann, H.; Pastor, P.; and Schaal, S. 2013. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation* 25(2):328–373.

Metta, G.; Sandini, G.; Vernon, D.; Natale, L.; and Nori, F. 2008. The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, 50–56. ACM.

Montesano, L.; Lopes, M.; Bernardino, A.; and Santos-Victor, J. 2008. Learning object affordances: from sensory-motor coordination to imitation. *IEEE Transactions on Robotics* 24(1):15–26.

Nicolescu, M. N., and Mataric, M. J. 2003. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 241–248. ACM.

Pairet, È.; Hernández, J. D.; Lahijanian, M.; and Carreras, M. 2018. Uncertainty-based Online Mapping and Motion Planning for Marine Robotics Guidance. In *Intelligent Robots and Systems (IROS), 2018 IEEE/RSJ International Conference on*. IEEE.

Pastor, P.; Hoffmann, H.; Asfour, T.; and Schaal, S. 2009. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, 763–768. IEEE.

Rai, A.; Meier, F.; Ijspeert, A.; and Schaal, S. 2014. Learning coupling terms for obstacle avoidance. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, 512–518. IEEE.

Smith, C.; Karayiannidis, Y.; Nalpantidis, L.; Gratal, X.; Qi, P.; Dimarogonas, D. V.; and Kragic, D. 2012. Dual arm manipulation: A survey. *Robotics and Autonomous systems* 60(10):1340–1353.

Topp, E. A. 2017. Knowledge for synchronized dual-arm robot programming. In *AAAI Fall Symposium Series 2017*. AAAI Press.

Ude, A.; Nemec, B.; Petrić, T.; and Morimoto, J. 2014. Orientation in cartesian space dynamic movement primitives. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2997–3004. IEEE.

Zöllner, R.; Asfour, T.; and Dillmann, R. 2004. Programming by demonstration: dual-arm manipulation tasks for humanoid robots. In *IROS*, 479–484.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Using hierarchical expectations grounded in perception for reasoning about failures during task execution

Priyam Parashar€, Ashok K. Goel£ and Henrik I. Christensen€

€ Contextual Robotics Institute, CSE Department, UC San Diego, 9500 Gilman Drive, La Jolla, CA USA
£ Design & Intelligence Laboratory, College of Computing Georgia Institute of Technology, Atlanta, Georgia USA
[pparasha, hichristensen] @eng.ucsd.edu, ashok.goel@cc.gatech.edu

Abstract

Traditionally, meta-reasoning architectures for planning have used abstract representations of expectations about the states of the world. However, embodiment of meta-reasoning on a robot requires grounding the expectations in perception. We propose a dual encoding of expectations based in the concept of occupancy grids. We illustrate this encoding for the task of designing shapes by block placement on a tabletop.

Introduction

Agent architectures for meta-reasoning typically contain three levels or types of information processing: an object level that perceives the world and acts on it; a deliberative level that makes sense of observations of the world and plans actions on world, and a meta-level that monitors and controls the deliberative level through goals and strategies, failures and repairs, and learning and adaptation. Figure 1 outlines a basic general meta-reasoning architecture. (Cox 2005) provides a review of many meta-reasoning architectures; Cox & Raja (2011) provide a more recent anthology of projects on meta-reasoning.

In a meta-reasoning architecture, when an agent reasons about failures, it first generates expectations about the state of the world, then compares the observed state of the world with the expected state, next maps the discrepancy between expected and observed state (the failure) into one or more repairs at the deliberative level (Stroulia & Goel 1995; Murdock & Goel 2008). The recognition of a failure through a comparison of the expected state and the observed state can be challenging if the observations are made through low-level sensors and the expectations are encoded in terms of abstract knowledge representations. Meta-reasoning architectures sometimes use specialized procedures to address this problem (Stroulia & Goel 1999; Jones & Goel 2012). For example, the Augur system uses

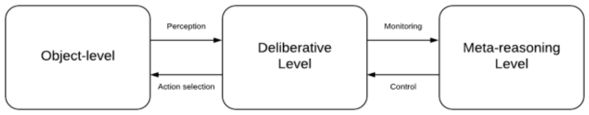


Figure 1. General Meta-reasoning Architecture

specially designed “empirical verification procedures” to map abstract knowledge representations with perceptual observations (Jones & Goel 2012).

For our work on robots capable of meta-reasoning (e.g., Parashar, Sheneman & Goel 2017), we seek a more general strategy for comparing expected and observed states and recognizing failures. In robotics, *seeing* sometimes garners more information than translating it in terms of the presence or absence of pre-defined symbols. Grounding the expectations into perception may have more transparency between the way a robot and the algorithm operating in its mind experience the world. Thus, we propose the use of *dual encodings* of expectations that include low-level visual encodings along with abstract representations.

In particular, in this article, we tackle the problem of designing shapes on a tabletop via block placements. Such a task design problem requires visual details beyond symbolic bindings to be actuated. Our approach uses the concept of *functional indexing in meta-reasoning* (Stroulia and Goel 1995) to encode domain knowledge from visual percepts into a *grid-map* which describes the form of physical placement of objects on the workspace. We propose a hierarchical representation for expectations where top-levels are defined by variables bindings and graph-based block-level symbol relationships, which can then be traced down into the hierarchy to find the grid-map encoding of expected block placement.

Our experimental setup uses the robot Baxter which is furnished with a library of hand-coded hierarchical task networks and annotated expectation databases to plan for drawing shapes. We create example failure cases to compare the meta-reasoning cycle which uses our expectations against one which only uses the symbolic-level.

The rest of the paper gives a quick introduction of relevant concepts and related literature. Next we explain the experimental setup and our methodology for extracting expectation annotations from an image feed. Finally, in the results section we present a qualitative assessment of our system for failure recovery where failures are induced by changing the environmental conditions to mismatch plan pre-conditions at different depths.

Background

We use Hierarchical Task Networks (Erol, Hendler and Nau 1994) to represent the plans for drawing shapes out of blocks. HTNs have two kinds of tasks: primitive actions or nonprimitive tasks. Each goal task is recursively decomposed into sub-tasks until a network of primitive actions remains, which is the base unit of execution, directly triggering some physical actuation in the agent. Nonprimitive tasks on the other hand are compound entities which can be further decomposed into sub-tasks. Each task has a set of pre-conditions and effects attached to it. Pre-conditions define in which state can a task be used to decompose a parent goal and effects depict how the task changes the state. In order to decompose a task into sub-tasks HTN uses the following rule: $\{\mathrm{t}\} \rightarrow \{\mathrm{t}', \mathrm{t}''\}$, if $\mathrm{pre}(\mathrm{t}') = \mathrm{pre}(\mathrm{t})$, $\mathrm{pre}(\mathrm{t}'') = \mathrm{eff}(\mathrm{t}')$ and $\mathrm{eff}(\mathrm{t}'') = \mathrm{eff}(\mathrm{t})$. In our system, each task is a hand-coded network which can be executed for drawing the symbol, for example, task **A** is a network which when executed forms the shape A on the table using blocks.

We know from the introduction section above that a meta-reasoner (figure 1) uses expectations to monitor agent's processes, records an execution trace and then reasons over it to find explanations for a failure. A complete meta-reasoning cycle would: (1) note ambiguities using expectations, (2) assess the reason for the unexpected scenario, and (3) reason over the plan trace and environment to guide a solution. Ambiguities are defined as mismatch between expected state description and encountered state during execution, defined by a *mismatch vector* which is a zero-vector of the same size as an expectation's symbolic description with mismatched variables assigned a 1. Finding explanations for ambiguities is a research area of its own and we are focusing on a very narrow range, i.e., unseen configurations of known objects. The current paper only addresses the first two stages of this pipeline for the outlined problem.

Related Work

Another framework relevant to meta-reasoning is that of goal-driven agents (Muñoz-Avila, et al. 2010) which uses expectations to monitor failures, and uses failures as an opportunity to learn new intermediate goals which can help with the overall task goal. Our work has some similarity with (Dannenhauer and Muñoz-Avila 2015), since they too use HTN plans annotated with expectations to conduct meta-level reasoning over their incomplete plans. However, their expectations are of a conceptual form, abstracted on top of environmental symbols.

(Stroulia and Goel 1995) use an explicit "structure-behavior-function" model in to assign blame to various parts of system's current design in a failure case. (Jones and Goel 2012) present "Empirical Verification Procedures" which ground all high-level concepts and axioms known to the agent in lower-level percepts of a video game. (Parashar, Sheneman and Goel 2017) combine meta-reasoning with reinforcement learning using purely visual form expectations. However, they still use symbolic descriptions or computerized descriptions of visual which simplifies the perception part of the problem.

In robotics, (Beetz, Mösenlechner and Tenorth 2010) and (Cox, et al. 2016) have used reasoning over abstracted concepts to help with low-level manipulation and task planning in robotics. However, all these methods use the symbolic layer to integrate reasoning with the environment.

Approach

Experimental Setup

We are using Mega Bloks$^{\mathrm{TM}}$ to draw shape and for simplicity will be referring to a single unit as a block. Our system considers two different shapes of blocks: 1x1 and 1x2; and supports two different colors: blue and red. Goals are communicated as strings naming the shape to be drawn. Each block's physical placement is described by two attributes, its orientation with respect to the table's axis and the location of its centroid in the workspace (figure 1). When blocks are recognized in an image they are indexed with a number starting at 0, e.g. $\mathrm{b}^0 = \{\mathrm{color},\ \mathrm{shape}\}$. To describe the placement of two blocks with respect to each other we use a graph-based format where $\mathrm{R}_{0,1}$ is a transform which when applied to the orientation and location of centroid of block b^0 would result in the centroid of block b^1. A 1x1 sized Mega Blok is of length 6.1 cm and width 6.1 cm, which we denote as l_b in the rest of the paper.

In order to codify the pre-conditions and effects of the HTN tasks we use a symbolic state descriptions which include: (a) $\mathrm{ob}_{\mathrm{grip}}$: a single-value set depicting the block in the gripper, φ if empty and $\{i, j\}$ if a block of color i and shape j is in the gripper, and (b) B: set of pairs depicting the required blocks and their availability. Other variables have been abstracted because they are not relevant to current discussion. Similarly, the only primitive action is: *place(b, x, y)* which subsumes smaller actions within it.

Hierarchical Representation of Expectations

The planning framework is still only using symbolic descriptions since we do not want the high-level planner to be bogged down by the details of the scene, however if a failure is noted we need access to a deeper knowledge-base which is encoded in our expectations. Hierarchical expectations are coded so that the 0th level has block-symbols describing the resultant composition effect an action and on the next or 1st level, we have cropped visual *grid-maps* centered on each block to capture a locally detailed description of its placement. Each grid-map is of length $3l_b$x$3l_b$ to include the block and some of the surrounding context. The plans are annotated with expectations at the primitive action level and backtracked to the goal task-level by assigning the parent task the expectation of last primitive action in its decomposition.

The expectation extractor uses a top-view image feed of workspace, via the robot's eye-in-hand setup, to extract expectations associated with each stage of the hierarchical task network. The block-level description of a symbol is extracted by performing HSV color-thresholding for blob detection on the tabletop view of the symbol under-construction. Once a colored blob is found, its shape is assigned by comparing blob-axis with l_b. Next, the visual expectation is a cropped centered on the centroid of the blob, and a quantized view of the form of the blob, i.e., a

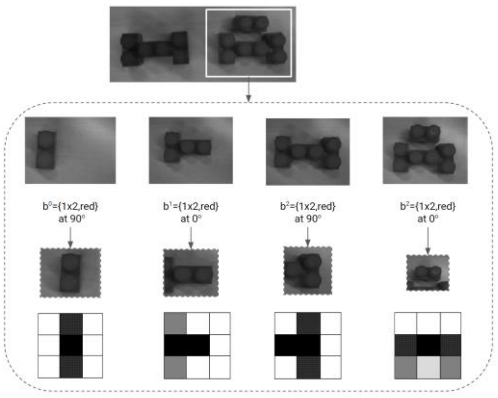


Figure 2. Depiction of Hierarchical Expectations, position of centroid removed from symbolic description for brevity. At the top, darker H shape is blue while lighter A is red.

grid-map is created. A grid-map is like an occupancy grid where the occupancy of a cell is decided based on color presence of the block on a uniformly colored background. The resolution of grid-map is $\frac{1}{2}l_b$.

It is to be noted that such a low-level description would require domain knowledge to be encoded since its form is tightly integrated with the goal of the problem itself. In the current implementation, the HTN plan is executed by an expert kinesthetically driving the agent to annotate the plan with resulting "ideal profile" of expectations. After the full execution, the expectation annotator creates two kinds of databases, one of annotated plans, denoted as P_a, and another one of the low-level grid-maps and symbolic expectations annotated by the causing action(s) which points back to the parent task itself, denoted by E_{db}. If multiple instantiations collide with the same expectation entry, then that expectation is annotated with all the corresponding action labels. The second database is key for a two-way communication between sensor information and plan knowledge even when symbol grounding fails during runtime.

Experimental Results

As one can see by looking at the state description, the failure can be of logical or physical kind. By physical we mean misplacement of gripper, wrong state of gripper, etc. This paper does not address these failures. In the rest of the paper when we explain our algorithm we are addressing only the logical failures, i.e., missing blocks, unexpected configuration of blocks, etc. We broadly classify the failures into two kinds, one where known entities are observed in an unseen configuration thus going against the explicit nature of pre-conditions, and one where unknown entities are observed breaking the planner's assumptions. We present here an example case where we create both kinds of failures by manipulating the environment. In this example, we have provided the agent with the plans for shape *A* and *H* (figure 2), using two 1x1 blocks for H rather than one 1x2 block. We use these shapes because they possess the kind of form similarities we want our algorithm to identify. We want to see if our expectations can help in creating connections between pieces of knowledge already stored in our database better than symbolic expectations. Next, the environment is modified to progressively make the failure more difficult for drawing the shape A. We replace required 1x2 block with another:

- Block of same shape but different color
- Set of two 1x1 blocks of same color
- Set of two 1x1 blocks of different color

The mismatch vector is used to identify the 1st instance of action in the invoked task which uses the mismatched entity, i.e., block in our case. Next, this action's expecta-

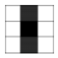
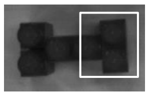
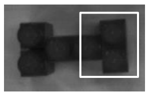
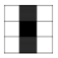
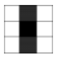
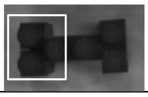
Type of Replacement	Affected Action-Exp Pair	Grid-map Match	Symbolic Match
1x2, red → 1x2, blue	b^0 = {1x2,red} at 90º 	b^4 = {1x2,blue} at 90º 	b^4 = {1x2,blue} at 90º 
1x2, red → 1x1, red + 1x1, red	b^0 = {1x2,red} at 90º 	None	None
1x2, red → 1x1, blue + 1x1, blue	b^0 = {1x2,red} at 90º 	b^2 = {1x1,blue} at 0º 	None

Table 1. Summary of Match results. The white square shows which block's resultant placement expectation in shape H was matched

tion is retrieved from P_a and a nearest-neighbor algorithm is invoked to find ranked matches from E_{ab}. We compare the entries retrieved by symbolic matching and grid-map matching to qualitatively assess the usefulness of our hierarchical expectation representation.

Results and Discussion

Our results are summarized in table 1 and compare the grid-map retrievals against symbol expectation matching. The most significant result is shown in row 3 where due to functional encoding of grid-maps its matches were able to search for a visual similarity of form unlike symbolic matching. For row 2, neither found a match since no shape uses {1x1, red} blocks in the current HTN plan library.

Our approach lends itself naturally to hybrid execution architectures where reactive learners manipulate raw-data and work in synchrony with deliberative planners which rely on some heuristic or some other form of domain knowledge. While it is easy to think of meta-reasoner as only an additional layer, its strength lies in enabling trading of valuable information across these two layers. It is this strength of meta-reasoner to form a global view which we believe will be a valuable addition to the long-term autonomy literature in robotics. Specifically, its across-event reasoning can augment the strength of episodic performance exhibited by reactive learners and task-oriented planners.

Conclusion

We presented in this paper dual encoding of expectations used in meta-reasoning to ground abstract representations of world states within perceptual encodings. We performed an experiment by changing our task environment to break pre-conditions coded for the required task and compared the visual expectations against the abstract expectations to see which form can help in recognizing failure as a first step towards failure recovery. Our results match our hypothesis: by encoding functional and visual form of the world within expectations, the meta-reasoner can make better connections within its knowledge base.

References

Beetz, M., L. Mösenlechner & M. Tenorth. 2010. CRAM - A Cognitive Robot Abstract Machine for everyday manipulation in human environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems.* 1012-1017.

Cox, M.. 2005. Field Review: Metacognition in Computation: A Selected Research Review. *Artif. Intell.* 169: 104-141.Cox, M. , Z. Alavi, D. Dannenhauer, V. Eyorokon, H. Muñoz-Avila & D, Perlis. 2016. MIDCA: A Metacognitive, Integrated Dual-Cycle Architecture for Self-Regulated Autonomy. *AAAI.* 3712-3718.

Dannenhauer, D & H. Muñoz-Avila. 2015. *Goal-Driven Autonomy with Semantically-Annotated Hierarchical Cases. Case-Based Reasoning Research and Development*, 88-103.

Erol, K., J. Hendler & D. Nau. 1994. HTN planning: Complexity and expressivity. *AAAI.* 1123-1128.

Jones.,J. & A. Goel. 2012. Perceptually grounded self-diagnosis and self-repair of domain knowledge. *Knowledge-Based Systems* 27: 281-301.

Muñoz-Avila, H., U. Jaidee, D. Aha & E. Carter. 2010. Goal-Driven Autonomy with Case-Based Reasoning. *Case-Based Reasoning. Research and Development,* pp. 228-241.

Parashar, P, B. Sheneman & A. Goel. 2017. *Adaptive Agents in Minecraft: A Hybrid Paradigm for Combining Domain Knowledge with Reinforcement Learning.* In Procs AAMAS-2017, pp. 86-100.

Stroulia, E, & A. Goel. 1995. Functional representation and reasoning for reflective systems. *Applied Artificial Intelligence* 9: 101-124.

Stroulia, E., & Goel, A. 1999. Evaluating Problem-Solving Methods in Evolutionary Design: The Autognostic Experiments. *Human-Computer Studies* 51:825-847, 1999.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Partial Policy Re-use in Connected Health Systems

Matthew Saponaro and Keith Decker

Abstract

We examine Probabilistic Partial Policy Reuse (PPR) for the purposes of developing tailored coaching strategies in the Coach-Trainee Problem (CTP). Policy reuse (PR) aims to improve a reinforcement learning agent by guiding exploration with past similar problems' learned policies. PPR extends probabilistic policy reuse that transfers only relevant parts of a policy for new problems. We explore PPR in the context of a human CTP where a coaching agent must develop a coaching strategy for the human trainee in order for the trainee to efficiently solve their problem (e.g. lose weight). In human CTPs, coach training data is limited because collecting too much data may annoy/discourage/harm the human trainee. In this paper, we present a decision tree-based algorithm, DT-partition, to identify partitions in the state space based on the problem's features, and also examine the effects of grouping problem meta-data (i.e. pruning the decision tree) on CTP performance. Particularly, we demonstrate that PPR improves task library generation and expert policy utilization compared with policy reuse.

Introduction

Wearable devices, such as smart watches, allow people to monitor their daily behaviors. Current population estimates of accelerometer-derived data indicate nearly 90% of US adults are insufficiently active and are sedentary for over half of all waking hours (55% or 7.7 hours/day) (Tucker, Welk, and Beyler 2011; Troiano et al. 2008; Matthews et al. 2008). Smart coaches, hosted/integrated with wearable devices, can improve physical activity by nudging their users to do exercise. Unfortunately, most commercial smart coaches only provide general intervention strategies and don't make use of the available information specific to their user. For example, Fitbit provides nudges 15 minutes prior to the end of each waking hour if the wearer does not walk more than 250 steps. Instead, an effective smart coach should take the context of the wearer into consideration and provide nudges when the wearer is receptive to the nudge. To create this tailored intervention strategy (i.e. determining when to nudge), a smart coach may learn when their user is receptive. We refer to learning the intervention strategy as the coach-trainee problem (CTP). For the purposes of this paper, we assume learning is done via reinforcement on the human responses. In order to learn the intervention strategy, the smart coach can probe the human user with a nudge and observe the behavioral response. Learning human behavior in this scenario is challenging because too many unsuccessful nudges may result in the smart coach being ignored or not worn. However, because there are similarities across humans, we can consider solving a single CTP by using information from different, previously solved CTPs (i.e. multi-task learning).

In this paper, we examine probabilistic policy reuse (PR) where past policies are used to guide the exploration of a new, never-before-seen problem. Unfortunately, as seen with many multi-task learning problem solutions, reusing past policies may yield negative transfer. In the human CTP, negative transfer may result with significant consequences (e.g. non-wear). PR attempts to avoid negative transfer by developing a core library of policies, then uses a policy from the library with a probability based on the utility it achieves annealed over time. In many multi-task learning scenarios, aspects of learning problems may be associated with a set of common features. In human coaching domains, humans with common traits often share similar behaviors. For example, many graduate students stay up late, while most preschoolers go to bed early. In a reinforcement learning context, these problem features may affect the underlying reward function or transition model of the problem. For the purposes of this paper, we assume that our CTP problem features are the trainee's accessibility to the gym, park, and unhealthy food options which are known contributors to a person's exercise habits. The set of problem features in a particular multi-task learning scenario are specific to the domain. For example, in a Pacman learning environment, the problem features may be the number and difficulty of the ghosts. In human coaching domains, parts of an intervention strategy may be useful to many different types of behavioral profiles but the full strategy may include some useless or harmful nudges to particular individuals. Through an indirect relationship, the solutions to learning individuals' coaching strategies are influenced by these problem features.

In this paper, we examine the effects of problem features on our CTP reinforcement learning problem solutions in order to more effectively develop smart coaches. First, we develop an algorithm to identify the subset of states controlled by the problem features (i.e. state space partitions) based on previously solved problems. Second, we develop a library of partial policies, where a partial policy is a policy for a subset of states. Lastly, we examine how to integrate the partial policy library into the exploration of a new problem. The key benefit of partial policy reuse is that it captures the most relevant parts of the policy, rather than using irrelevant parts of a whole policy that resulted in net positive transfer. Additionally, completed partial policies could be used immediately without learning the whole policy. Finally, the decision trees generated by our algorithm could be initially developed by a human expert rather than computed strictly from the source tasks as presented in this paper. A human expert (e.g. fitness coach) with domain specific knowledge (e.g. what types of individuals respond well to different intervention strategies) can effectively provide initial solutions. In the human coaching domain, initial human data may be scarce so a human expert can effectively jump start the proposed algorithms (i.e. build the decision trees for DT-partition).

The remainder of this paper is outlined as follows: In section 2, we discuss the related work done in multi-task learning. In section 3, we discuss our problem definition. In section 4, we introduce the specific problem that we are trying to address. In section 5, we discuss our partial policy reuse algorithms. In section 6, we describe our experimental setup, results, and analysis. Finally, we conclude our work and discuss our next steps in section 7.

Related Work

In many multi-task learning domains, meta-data regarding problem features may describe the types of problems previously learned (Sinapov et al. 2015; Rosman, Hawasly, and Ramamoorthy 2016). For example, in the coaching domain, occupation and BMI may influence a person's ability to do moderate exercise throughout the day (e.g. waitresses walk around the restaurant while at work whereas graduate students must sit in front of a computer all day). Sinapov et al. use multi-task meta-data to develop a M5 model tree to estimate the transfer benefit from source to target task (Sinapov et al. 2015). Policies of the highest estimated benefit source tasks were directly used for the targets. Unfortunately, their approach requires a significant amount of data (i.e. solved problems) to develop their trees which makes it infeasible in the human coaching domain. In this paper, we attempt to use the problems' meta-data to a develop decision tree for each state to identify which parts of the solutions' (i.e. partial policies) state space are controlled by which problem features and thus useful for transfer. In order to address the training data issue, we extend Fernandez et al.'s work (Ferndez, Garca, and Veloso 2010). Fernandez et al. develop a policy reuse algorithm for guiding policy exploration for reinforcement learning problems; however, Fernandez et al.'s problems do not have meta-data regarding problem features. Particularly, policy reuse uses a Boltzmann exploration strategy on past solved policies weighting the use of a policy based on how well it did to solve the new problem. In this paper, we address a common issue found in both of these works—irrelevant/not useful parts of the source tasks' policies are being transferred. Because our state space partitioning algorithm identifies relevant parts of the solution based on the problem features, we improve transfer effectiveness. Furthermore, we extend Fernandez et al.'s Boltzman exploration to partial policies which then improves training data utilization. Finally, we extend Fernandez et al.'s algorithm to identify the core policy library to identify a core partial policy library.

In reinforcement learning scenarios, deep architectures typically require substantial data to train (Glatt, Da Silva, and Costa 2016). In human coaching domains, training data is limited. Furthermore, existing deep architectures require completed solutions prior to transfer. Deep architectures like ADAAPT and Actor-mimic network use neural networks to determine how to do transfer; however, decision trees provide meaningful descriptions that a domain expert (e.g. human fitness coach) may utilize to jump start agent learners (Parisotto, Ba, and Salakhutdinov 2015; Rajendran et al. 2015). Additionally, attribution of problem features to solutions can be easily identified and later used for feedback to the human fitness coaches to update then guide coaching agent learners' domain knowledge (both jump start and online).

In this paper, we make several assumptions regarding the human model that are commonly found in the literature (Saponaro, Wei, and Decker 2017; Muntaner, Vidal-Conti, and Palou 2015; Pavel et al. 2015). The goal for this coaching system is to provide a tailored intervention strategy for each trainee since general strategies tend to do poorly (op den Akker, Jones, and Hermens 2014). Trainees in this system are assumed to be either in the contemplation, preparation, or action phases of the trans-theoretical model of behavior change where trainees are thinking about and/or attempting to make a behavioral change by educating or taking action in their exercise regimen (Prochaska 2013). Furthermore, we provide trainees with just-in-time and adaptive nudges precisely when trainees are susceptible for change (in the moment), rather than scheduled. We consider trainees persisting in free-living conditions, as opposed to laboratory controlled environments (Suay and Chernova 2011). Furthermore, in this paper, we only consider when to provide nudges for activity. We do not consider the intention, content, or representation of the nudge due to the complexity of the system (op den Akker et al. 2015). Instead of modeling the fine-grained detail such as GPS location and heart rate, we abstract the human state into general categories that are meaningful across trainees, but still relevant and useful. For example, numerical GPS location would be categorized meaningful contexts such as park and gym. In this paper, we consider the impulse model proposed by Hoffman et al. (Hofmann, Friese, and Strack 2009). In Hoffman et al., humans were effectively modeled by blending a habit MDP and impulse MDP together. Particularly, trainees will either react or not to nudges to do exercise and learn the correct evaluations of states due to being nudged. Furthermore, we assume our trainees follow the markov assumption; though, a semi-markovian model can be studied to improve accuracy (Pavel et al. 2015), we chose to simplify our human models to an MDP in order to compare our work with the other machine learning literature.

Problem Definition

We demonstrate our contributions in a multi-task reinforcement learning scenario as described by Sinapov et al. (Sinapov et al. 2015). We assume the underlying problem is represented as a Markov Decision Process (MDP). An MDP is a tuple, $\langle S, A, T, R\rangle$, where S is the set of states $S = \{s_1, s_2, s_3, \ldots, s_n\}$, A is the set of actions, T is the stochastic transition function $T : S \times A \times S \to \mathbb{R}$, and R is the reward function $R : S \to \mathbb{R}$. The agent learner does not have access to T and R. We assume our reinforcement learning problems have the same domain, $D = \langle S, A, T\rangle$ and differ only by reward function. A particular reinforcement learning problem is defined as $\Omega_i = \langle D, R_i\rangle$. Each Ω_i is associated with a feature vector, $F_i = [f_1, f_2, \ldots f_m]$, such that there is an unknown and stochastic relationship between F_i and R_i.

An episode, k, of Ω_i starts with the agent located in an initial state and ends when the agent reaches a goal state (terminal) or after completing H steps. In this paper, the agents' goal is to maximize the expected average reinforcement per episode, W. This goal definition is taken from the defining policy reuse paper (Fernndez, Garca, and Veloso 2010):

$$W = \frac{1}{K}\sum_{k=0}^{K}\sum_{h=0}^{H}\gamma^h r_{k,h} \tag{1}$$

where γ is the discount factor and $r_{k,h}$ is the reward signal received from step h in episode k. In our specific application area, it is important to learn and receive positive reward early since the human may become annoyed quickly. In order to achieve its goal to optimize W, the agent learner must develop a policy $\pi : S \to A$ the dictates what action to be taken in each state. In this paper, we assume there are a set of source tasks, $T_{\text{source}} \subset \Omega$, that have been completely solved (i.e optimal policy) and a set of target tasks that have not been solved $T_{\text{target}} = \Omega - T_{\text{source}}$. In this paper, we examine developing the policy through Q-learning, though any learning algorithm to create a policy could be implemented.

The goal of Policy Reuse is to use solutions of previously solved tasks, to bias the exploration when learning the action policy of a new task in the same domain (Fernndez, Garca, and Veloso 2010). In this paper, we extend Fernandez and Velosa's policy reuse library and exploration bias strategy for partial policies. Particularly, we aim to build a partial policy library that builds a policy library for different partitions of the state space. We formally define the partial policy library as $L = \{L_{S_{p_1}}, L_{S_{p_2}}, L_{S_{p_3}}, \ldots, L_{S_{p_l}}\}$ where $L_{S_{p_i}}$ contains a library of core partial policies associated with the states $S_{p_i} \subset S$. We note that $S_{p_i} \in S_p$ is a partition of the state space and that all partitions have unique states (i.e. $S_{p_i} \cap S_{p_j} = \emptyset, \forall i \neq j$). The contributions of this paper explore how to effectively create and use this partial policy library (i.e. PPR). Particularly, we develop a library of relevant partial policies, $L_{S_{p_i}} = \{L_1^{S_{p_i}}, L_2^{S_{p_i}}, \ldots, L_\ell^{S_{p_i}}\}$ where $L_\ell^{S_{p_i}}$ is the set of core partial policies relevant to agent group ℓ in partition S_{p_i}. Later in the paper, we describe how to partition the state space (i.e. determine S_p) based on F for the purposes of identifying related solution parts (i.e. DT-partition). We then describe how to build a core partial policy library, L.

Algorithm 1 DT-Partition

```
 1: procedure DT-PARTITION($T_{\text{source}}, S,$)
 2:     for $s_i \in S$ do
 3:         $dt_i \leftarrow$ buildTree $(s_i, T_{\text{source}})$
 4:     end for
 5:     $D \leftarrow \emptyset$                    ▷ initialize state partition trees
 6:     $S_r \leftarrow S$                          ▷ unpartitioned states
 7:     $S_p \leftarrow \emptyset$                  ▷ initialize state space partitions
 8:     d = 0                                        ▷ initialize partition index
 9:     while $S_r \neq \emptyset$ do
10:         $s_k \leftarrow S_r$.pop ()
11:         $S_{p_d} \leftarrow S_{p_d} \cup \{s_k\}$      ▷ initialize partition $S_{p_d}$
12:         $D \leftarrow D \cup \{dt_k\}$                 ▷ initialize partition's DT
13:         for $s_i \in S_r$ do
14:             if DT-equal($dt_i, dt_k$) then
15:                 $S_{p_d} \leftarrow S_{p_d} \cup \{s_i\}$   ▷ add $s_i$ to partition
16:                 $S_r = S_r - \{s_i\}$
17:             end if
18:         end for
19:         $S_p \leftarrow S_p \cup \{S_{p_d}\}$
20:         $d = d + 1$                                  ▷ increment partition index
21:     end while
22:     return $S_p, D$
23: end procedure
```

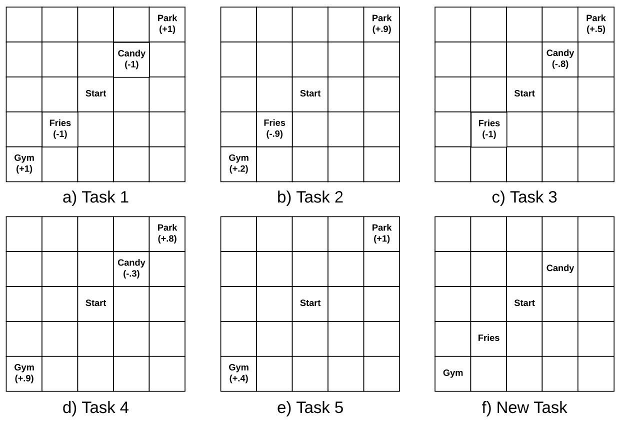


Figure 1: Grid-based CTP where a) through e) are source tasks and f) is the target task.

Grid-CTP

We demonstrate the effectiveness of partial policy reuse in a grid-based coach-trainee problem (CTP) domain where a coach must learn where an individual is most likely to do exercise while not being distracted by their surroundings.

In our grid-based coach-trainee problem, the underlying MDP can be modeled such that the states, S, are the grid locations that the trainee can be located, the actions are the nudged direction that the trainee should go, the transition model represents the trainee's likeliness to follow the coach's nudge, and the reward function, R, is the health benefits achieved by going to a particular location. All trainees are assumed to have the same transition model—follow the coach's advice 90% of the time and wander randomly the other 10%. The reward function's are different per problem and related to the problem features. In this paper, we examine features related to the trainee's surrounding location. Particularly, we examine the following trainee's accessibility features: park, gym, candy store, and a fry shop. Candy shops and fry shops are distractions to the health benefits achieved at parks and gyms. In figure 1, we show 6 grid-CTPs. Each grid-CTP have different problem features. For example, in figure 1 a), the trainee has access to the gym, park, fry shop, and candy store, but in figure 1 b) the trainee doesn't have access to a candy store. Due to the variance in individual trainees, we assume rewards take a uniform distribution where health benefits are between 0 and 1 and distractions are between -1 and 0. If a trainee has no access to a community utility (e.g. gym), then no reward is given for that state. For an example, in figure 2 d), the trainee has no access to a gym. An episode represents a day of the trainee and the trainee may only go to one community utility; thus, rewarding states are terminal. We assume there are only 18 steps per episode representing that a person only has 18 waking hours per day. Furthermore, we assume we have a collection of previously solved CTPs where the coach policy is complete. In this paper, we attempt to use this past collection of coach policies to quickly learn a new trainee's coach's policy.

We make several assumptions regarding CTP that do not affect the contributions of this paper. Particularly, we assume the transition model is fixed meaning that the trainee is not changing any behaviors due to incomplete or incorrect perceptions of the world. For example, in the full-CTP, a trainee may initially highly value eating french fries because they taste good, but when the coach indicates french fries are unhealthy, the trainee would update its views on french fries and avoid them. Furthermore, the coach does not model the trainee's readiness to receive an intervention. For example, a trainee may be busy in the morning and unwilling to do exercise, and therefore ignore whatever the coach says. Finally, we limit the problem features in this paper to the accessibility of different physical locations. In the full CTP, problem features include occupation, body mass index (BMI), age, social status, community utilities, etc. Though we do not examine the full problem, the algorithms in this paper can generalize to the full problem.

Partial Policy Reuse

In policy reuse, a source task policy is used to guide the exploration when solving a target task. We argue that the source task's full policy may include irrelevant/harmful parts in the target's problem. For example, all athletes may be susceptible to nudges in the morning; however, some find it beneficial to do light exercise at night, while other athletes find it exhaustively harmful. In this example, there may be a partition for the day coaching strategies and a partition for the night coaching strategies. Additionally, we argue that all source tasks may not be relevant to the target task's solution. For example, runners like to run at night because the air is cool, but weight lifters do not like to lift weights at night because the gym is dirty from being used during the day. In this example, using a coaching strategy for a jogger on a runner may make sense since both like to do exercise at night, but using the jogger coaching strategy for a weight lifter may result in negative transfer. Partial policy reuse aims to mitigate these issues by determining what are the relevant parts of the solution to transfer (i.e. DT-partition) and what are the relevant source task solutions to use when transferring these partial solutions (i.e. partial-policy library).

DT-Partition

This section explains the DT-partition algorithm. DT-partition aims to partition the state space based on the similarity of solutions for problems with the same features in order to identify parts of solutions that are relevant for transfer. We define a state space partition as a set of states $S_{p_i} \subset S$ such that a single partition has no overlap of states with any other partition. The following DT-partition algorithm partitions the states based on the agreement of the source task policies. See algorithm 1.

Algorithm 2 DT-Compare

```
 1: procedure DT-EQUAL($dt_i, dt_j$)
 2:     if nodeType $(dt_i) \neq$ nodeType $(dt_j)$ then
 3:         return $False$                    ▷ leaf vs. internal node
 4:     else if isLeaf $(dt_i)$ then
 5:         return $dt_i$.tasks $==$ $dt_j$.tasks
 6:     else
 7:         if $dt_i$splitAttribute $\neq$ $dt_j$.splitAttribute then
 8:             return $False$
 9:         else
10:             isEqual $\leftarrow$ $True$
11:             for $chld \in$ children do
12:                 subEql $\leftarrow$ DT-equal $(dt_i$.chld, $dt_j$.chld$)$
13:                 isEqual $\leftarrow$ isEqual && subEql
14:             end for
15:             return isEqual
16:         end if
17:     end if
18: end procedure
```

For each state, s_i, develop a decision tree, dt_i, using $\Omega_t \in T_{\text{source}}$ as the instances such that F_t are the features and $\pi_{\Omega_t}(s_i)$ is the class value. For the purposes of this paper, we use the C4.5 decision tree algorithm. Then, group states together (i.e. label as a partition) such that the decision trees are equivalent in structure. We assert that two decision trees are equivalent if the internal nodes have the same split points and the source tasks that reach the leaves are the same. See algorithm 2. We note that the leaf classifications may be different across trees, but as long as the instances that reach the node (i.e. Ω) and their class values are in agreement (i.e. $\pi_{\Omega_t}(s_i) = \pi_{\Omega_u}(s_i)$), we consider the trees equivalent.

In our experiments section, we analyze how pruning the tree structure (i.e. decreasing number of partitions) effects transfer learning outcome. We note that if a tree were completely pruned, the algorithm would produce a single partition equivalent to S and thus replicate policy reuse.

Partial-Policy Library

The partial policy library consists of relevant core policies for each partition of the state space for the purposes of effectively transferring knowledge. We describe how to develop a partial policy library in algorithm 3.

Algorithm 3 Build-Partial-Policy-Library

```
 1: procedure BUILD-PP-LIBRARY($T_{\text{source}}, S, \theta$)
 2:     $L \leftarrow \emptyset$
 3:     $S_p, D \leftarrow$ DT-partition $(T_{\text{source}}, S)$
 4:     for $i \in \{0, \ldots, |S_p|\}$ do
 5:         $L_{S_{p_i}} \leftarrow \emptyset$
 6:         for $\ell \in D_i$.leaves do
 7:             $L_l^{S_{p_i}} \leftarrow$ add-core-PP $(S_{p_i}, \ell.\Omega, \theta)$
 8:             $L_{S_{p_i}} \leftarrow L_{S_{p_i}} \cup \{L_l^{S_{p_i}}\}$
 9:         end for
10:         $L \leftarrow L \cup \{L_{S_{p_i}}\}$
11:     end for
12: end procedure
```

In our algorithm, we partition the state space using a partitioning algorithm like the one previously discussed. For each partition S_{p_i}, we use the corresponding partition's decision tree structure, D_i, to determine the source tasks for each relevant group ℓ. Instances at each leaf in D_i are considered within the same relevant source task group. In our experiments described later, we examine pruning the tree's structure to show the effects of available source tasks and relevancy for transfer.

As described in algorithm 5, using the relevant source tasks at each leaf, $\Omega_\ell^{S_{p_i}}$, we build the core partial library for $L_\ell^{S_{p_i}}$. For a particular source task, Ω_m, we add its corresponding policy, π_m, to the library if the percent increase in expected performance (i.e. $\hat{w}_m$) of using its policy compared with the expected performance of the best policy in the library is greater than a threshold, θ. In order to estimate the W, we use the task's Q-values. It is non-trivial to directly compute the partial policies expected performance because the performance is dependent on the full policy.

We note that when we fully prune the tree's structure when determining source task selection, all source tasks are used. Furthermore, when we prune both the state space partition tree and the source task selection tree, we reduce our partial policy reuse algorithms to the defining policy reuse algorithm as defined in (Fernndez, Garca, and Veloso 2010). Furthermore, though we present a decision tree based algorithm that uses agreement between policies, other optimizations that better partition the state space and identify relevant source tasks may be utilized.

Algorithm 4 Ω_{new} Learning from the partial policy library

```
 1: procedure LEARN($\Omega_{\text{new}}, L, K, H, v, \Delta\tau$)
 2:     $Q_{\text{new}}(s,a) = 0, \forall s \in S, a \in A$
 3:     $U_{S_{p_i},n} = 0, \forall i : S_{p_i} \in S_p, \forall n : \pi_n^{S_{p_i}} \in L_\ell^{S_{p_i}}$
 4:     $W_{s_{p_i},n} = 0$
 5:     $\tau = 0$
 6:     for k = 1 to $K$ do
 7:         $s \leftarrow s_{\text{start}}$
 8:         $\phi \leftarrow \phi_{\text{init}}$
 9:         $\pi_{S_{p_i}} \leftarrow \pi_{S_{p_i},n}$ with probability $\frac{e^{\tau \cdot W_{s_{p_i},n}}}{\sum_o e^{\tau \cdot W_{s_{p_i},o}}}$
10:         $W_k = 0$
11:         for h = 1 to $H$ do
12:             $S_{p_i} \leftarrow S_{p_k}$ s.t. $s \in S_{p_k}$
13:             With probability $\phi, a = \pi_{S_{p_i}}$
14:             With probability $1-\phi, a = \epsilon$-greedy $(\Omega_{\text{new}})$
15:             $s' \leftarrow$ next state, $r_{k,h} \leftarrow$ reward signal
16:             $W_k \leftarrow W_k + \gamma^h \cdot r_{k,h}$
17:             update $Q_{\text{new}}$ and $\pi_{\text{new}}$
18:             $\phi \leftarrow \phi \cdot v$
19:             $s \leftarrow s'$
20:         end for
21:         $\tau \leftarrow \tau + \Delta\tau$
22:         $U_{S_{p_i},n} \leftarrow U_{S_{p_i},n} + 1, \forall S_{p_i} \in S_p$
23:         $W_{s_{p_i},n} \leftarrow \frac{W_{s_{p_i},n} \cdot (U_{s_{p_i},n} - 1) + W_k}{U_{s_{p_i},n}}, \forall S_{p_i} \in S_p$
24:     end for
25: end procedure
```

Algorithm 5 Add Core Partial Policies

```
 1: procedure ADD-CORE-PP($S_{p_i}, \Omega_l^{S_{p_i}}, \theta$)
 2:     $L_l^{S_{p_i}} \leftarrow \emptyset$
 3:     for $\Omega_m \in \Omega_l^{S_{p_i}}$ do
 4:         $\pi_m \leftarrow \Omega_m.\pi$
 5:         $\hat{W}_{\pi_m} \leftarrow \sum_{s_j \in S_{p_i}} Q_m(s_j, \pi_m(s_j))$
 6:         $\hat{W}_{\text{best}} = \max_{\pi_k \in L_l^{S_{p_i}}} \sum_{s_j \in S_{p_i}} Q_m(s_j, \pi_k(s_j))$
 7:         if $\hat{W}_{\pi_m} - \hat{W}_{\text{best}} > \theta \left|\hat{W}_{\text{best}}\right|$ then
 8:             $L_l^{S_{p_i}} \leftarrow L_l^{S_{p_i}} \cup \{\pi_m\}$
 9:         end if
10:     end for
11:     return $L_l^{S_{p_i}}$
12: end procedure
```

Partial-Policy Reinforcement Learning

In this section, we describe how an agent may use the partial policy library to learn a policy for a new task Ω_{new}. See algorithm 4. Partial Policy reinforcement learning extends Policy Reuse reinforcement learning by simultaneously using multiple partial policies,$\pi_{S_{p_i}}$, to guide the agent learner for a particular episode, k.

For a particular episode, k, an agent chooses a partial policy to execute in each partition, S_{p_i}, of the state space. A partial policy, $\pi_{S_{p_i},n} \in L_\ell^{S_{p_i}}$ is chosen from the library based on the agent learner's performance, W. A particular partial policy is likely to be used when the agent has good performance and unlikely to be used when the agent has poor performance with respect to other partial policies in $L_\ell^{S_{p_i}}$. In the proposed partial policy reinforcement learning algorithm, the partial policy's individual performance, $W_{S_{p_i},n}$, is effected by the combined performance of all partial policies; however, $W_{S_{p_i},n}$ will eventually converge to the true evaluation after collecting enough data. We believe this problem of attributing individual partial policy performance is an open area for researchers to address in order to improve partial policy usages. For a particular step, h, the agent can either use its own policy, π_{new}, or use the expert partial policy, $\pi_{S_{p_i}}$. When the agent learner uses the expert partial policy, the agent passively learns and updates its Q-table based on the expert's action and the reward signal the learner received. The learning agent anneals the temperature parameter, ϕ, that controls the probability of using the expert's policy with respect to the number of steps taken. Additionally, the agent learner anneals the reuse parameter, τ, over the episodes and increases the likelihood to choose the best partial policies based on their performances.

In domains with limited source tasks, our partial policy library algorithm may result in state partition libraries (i.e. $L_\ell^{S_{p_i}}$) that do not have any expert policies. In this particular scenario, either the tree used to generate the libraries should be pruned to impose a larger source task library or the agent learner can simply use its own policy without the guidance of an expert.

Experiments

We demonstrate the effectiveness of our algorithms in the Grid-CTP domain. Particularly, we demonstrate the effects of our state space partitioning algorithm, DT-partition, on coach learning performance and the effects of the relevance of source tasks on our coach's performance. The algorithms presented in this paper effectively target two questions: 1) How much of the state space can be transferred in order to effectively guide the new learner? 2) Which tasks are relevant enough to be used when guiding the new learner? In order to demonstrate that our algorithms address these questions, we present the following experiments. The first experiment prunes the trees used to make the state partitions in DT-partition. By pruning these trees, the partial policies become larger (i.e. $|S_{p_i}|$ increases). The second experiment prunes the decision trees after creating the state space partitions; thus, increasing the number of the expert policies in the source task library.

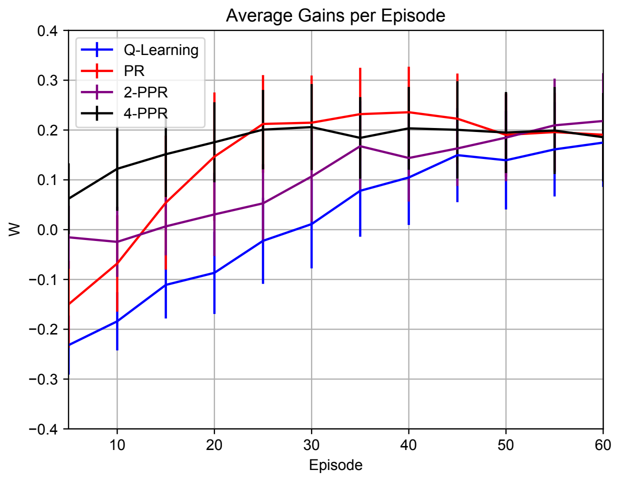


Figure 2: PPPR state space partition decision tree was pruned to have a height of 2 (i.e. 2-PPR) and 4 (i.e. 4-PPR)

We generated 160 source tasks—10 repetitions of all combinations of our 4 Grid-CTPs' problem features (i.e. accessibility to gym, park, candy store, fry shop), where the reward functions of each task take on a uniform distribution between 0 and 1 for states with health benefits (i.e. gym and park) and 0 and -1 for health distractions (i.e. candy and fries). We then use these 160 source tasks as experts to guide a coach learner to solve a new task. In order to efficiently use these source tasks, we first build the partial policy library as described above in algorithm 3. In order to add a partial policy to the library, the partial policy needs to have a 10% performance increase (i.e. $\theta = .1$). We then use the partial policy library to execute a coach learner on a new problem as described in algorithm 4. The new problem has accessibility to the gym, fry shop, and candy store, but no accessibility to the park. Our coach learner's initial ϕ is 1, and cools down by 1% each step (i.e. $v = .01$). The coach expert preference temperature parameter, τ increases linearly by 1 unit every episode (i.e. $\Delta\tau = 1$); thus $\frac{1}{\tau}$ decreases and prefers better experts in future episodes. We execute the learning phase for the agent coach 100 times and take the average performance across 100 runs for each episode.

Increasing the State Partition Size

In this experiment, we build the full decision trees for each state used to partition the state space, then prune them to heights of 0, 2, and 4 before partitioning the state space in algorithm 1. By pruning the trees, we decrease the entropy of agreement of policies in the leaves. We note that when we prune the trees to a height of 0, the algorithm reduces to policy reuse. We show the performance of the coach learning using partial policy reuse when the heights of the trees are 2 (i.e. 2-PPR), and 4 (i.e. 4-PPR) in figure 2.

We see that partial policy reuse, when building the full tree, does best compared against standard Q-learning and Policy Reuse. Interestingly, the relationship between the height of the tree and the performance is not linear. Particularly, the decision trees with height 2 do worse than when the heights are 0 and 4. We hypothesize that the decision trees of height 2 perform poorly with respect to the other heights because the useful amount of knowledge per partition was low. Particularly, we believe there were too many decisive states (i.e. parts of trajectories that led to reward), yet not enough useful knowledge gained from the experts in these partitions.

Increasing the Quantity of Source Tasks in Library

In this experiment, we partition the state space using algorithm 1; however, after we've determined the state space partition, we prune the decision trees to heights of 0, 2, and 4 before developing the core partial policy library. For a particular partition, pruning the tree increases the number of source tasks used in the partial policy library for that partition. When plotting the results of average performance per episode, We get similar results as shown in figure 2 where the fully developed tree performs best, while the height of 2 tree performs only better than Q-learning. We hypothesize the that partitioning algorithm may not be yielding effective partial expert policies in some of the partitions.

Conclusion

In this paper, we demonstrate the effectiveness of probabilistic partial policy reuse in the CTP domain. In human learning domains where training data is limited and costly, agents need to quickly learn a solution in order to not be seen as annoying or irrelevant (i.e. low time to threshold). Partial policy reuse allows agent learners to achieve this by partitioning the state space and using relevant partial policies to solve a problem. In this paper, we present novel algorithms to partition the state space, develop a core partial policy library, and use a partial policy library while learning. Our algorithms allow for agent learners to learn parts of the state space and become partial experts (i.e. source tasks) without completely solving the underlying problem. The partial policy reuse algorithm presented in this paper trades off both the amount of transferred content and the availability of source tasks with the relevance of source tasks. In this paper, we examine decision tree based solutions because domain experts (i.e. fitness coaches) may be able to manually provide a structure based on their expert opinion. For example, in the CTP domain, a physical activity coach may notice that occupation is a strong indicator of group behavior in physical activity. Particularly, nurses walk often during the day, so nudges may be useless during the day, but for graduate students, nudges may be useful during the day. Though we develop partial policies, individual partial policy performance attribution is an open question. In realistic human learning systems such as CTP, a better attribution design may be needed. The work presented in this paper attempts to improve learning efficiency in order to create tailored interventions strategies in the human coaching domain; however, our work can also be applied to other multi-task learning domains.

References

Fernndez, F.; Garca, J.; and Veloso, M. 2010. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems* 58(7):866 – 871. Advances in Autonomous Robots for Service and Entertainment.

Glatt, R.; Da Silva, F. L.; and Costa, A. H. R. 2016. Towards knowledge transfer in deep reinforcement learning. In *Intelligent Systems (BRACIS), 2016 5th Brazilian Conference on*, 91–96. IEEE.

Hofmann, W.; Friese, M.; and Strack, F. 2009. Impulse and self-control from a dual-systems perspective. *Perspectives on Psychological Science* 4(2):162–176.

Matthews, C. E.; Chen, K. Y.; Freedson, P. S.; Buchowski, M. S.; Beech, B. M.; Pate, R. R.; and Troiano, R. P. 2008. Amount of time spent in sedentary behaviors in the united states, 2003–2004. *American journal of epidemiology* 167(7):875–881.

Muntaner, A.; Vidal-Conti, J.; and Palou, P. 2015. Increasing physical activity through mobile device interventions: A systematic review. *Health informatics journal* 1460458214567004.

op den Akker, H.; Cabrita, M.; op den Akker, R.; Jones, V. M.; and Hermens, H. J. 2015. Tailored motivational message generation: A model and practical framework for real-time physical activity coaching. *Journal of biomedical informatics* 55:104–115.

op den Akker, H.; Jones, V. M.; and Hermens, H. J. 2014. Tailoring real-time physical activity coaching systems: a literature survey and model. *User modeling and user-adapted interaction* 24(5):351–392.

Parisotto, E.; Ba, J. L.; and Salakhutdinov, R. 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

Pavel, M.; Jimison, H. B.; Korhonen, I.; Gordon, C. M.; and Saranummi, N. 2015. Behavioral informatics and computational modeling in support of proactive health management and care.

Prochaska, J. O. 2013. Transtheoretical model of behavior change. In *Encyclopedia of behavioral medicine*. Springer. 1997–2000.

Rajendran, J.; Prasanna, P.; Ravindran, B.; and Khapra, M. 2015. Adaapt: A deep architecture for adaptive policy transfer from multiple sources. *arXiv preprint arXiv:1510.02879*.

Rosman, B.; Hawasly, M.; and Ramamoorthy, S. 2016. Bayesian policy reuse. *Machine Learning* 104(1):99–127.

Saponaro, M.; Wei, H.; and Decker, K. 2017. Towards learning efficient intervention policies for wearable devices. In *Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2017 IEEE/ACM International Conference on*, 298–299. IEEE.

Sinapov, J.; Narvekar, S.; Leonetti, M.; and Stone, P. 2015. Learning inter-task transferability in the absence of target task samples. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, 725–733. International Foundation for Autonomous Agents and Multiagent Systems.

Suay, H. B., and Chernova, S. 2011. Effect of human guidance and state space size on interactive reinforcement learning. In *2011 Ro-Man*, 1–6. IEEE.

Troiano, R. P.; Berrigan, D.; Dodd, K. W.; Masse, L. C.; Tilert, T.; McDowell, M.; et al. 2008. Physical activity in the united states measured by accelerometer. *Medicine and science in sports and exercise* 40(1):181.

Tucker, J. M.; Welk, G. J.; and Beyler, N. K. 2011. Physical activity in us adults: compliance with the physical activity guidelines for americans. *American journal of preventive medicine* 40(4):454–461.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Multi-Fidelity Model-Free Reinforcement Learning with Gaussian Processes

Varun Suryan, Nahush Gondhalekar, Pratap Tokekar
Virginia Tech, USA

Abstract

We study the problem of Reinforcement Learning (RL) using as few real-world samples as possible. A naive application of RL can be inefficient in large and continuous state spaces. We present a Multi-Fidelity Reinforcement Learning (MFRL) model-free algorithm that leverages Gaussian Processes (GPs) to learn the optimal policy in the real world. In the MFRL framework, an agent uses multiple simulators of the real environment to perform actions. With increasing fidelity in a simulator chain, the number of samples used in successively higher simulators can be reduced. By incorporating GPs in the MFRL framework, further reduction in the number of learning samples can be achieved as we move up the simulator chain. We examine the performance of our algorithm through simulations and through real-world experiments for navigation with a ground robot.

Introduction

Reinforcement learning (RL) allows an agent to learn directly from an environment without relying on exemplary supervision or complete models of the environment (Sutton and Barto 1998). Recently, there has been a significant development in RL applied to learning policies for robots (Mnih et al. 2015; LeCun, Bengio, and Hinton 2015; Silver et al. 2016). A major limitation of using RL for learning with robots is the need to obtain a large number of training samples. Obtaining a large number of real-world samples can be expensive and potentially dangerous. In particular, obtaining negative samples may require the robot to collide or fail, which is undesirable. The overall goal of our work is to reduce the number of real-world samples required for learning optimal policies. In this paper, we show how to leverage one or more simulators along with real-world samples to learn a policy for a robot.

We build on the Multi-Fidelity Reinforcement Learning (MFRL) algorithm from (Cutler, Walsh, and How 2015). MFRL leverages multiple simulators with varying fidelity levels to minimize the number of real-world (*i.e.*, highest fidelity simulator) samples. The simulators, denoted by $\Sigma_1, \ldots, \Sigma_d$, have increasing levels of fidelity with respect to the real environment. For example, Σ_1 can be a simple simulator that models only the robot kinematics, Σ_2 can model the dynamics as well as kinematics, and the highest fidelity simulator can be the real-world (Figure 1).

This work has been funded by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation-sponsored industry/university cooperative research center (I/UCRC) under NSF Award No. IIP-1161036 along with significant contributions from C-UAS industry members.

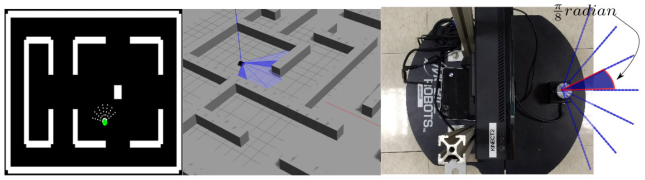


Figure 1: MFRL framework: The first simulator captures only grid-world movements of a point robot while the second simulator has more fidelity modeling the physics as well. Control can switch back and forth between the simulators and real environment which is the third simulator in the chain. We use the Python-based simulator Pygame as Σ_1, Gazebo as Σ_2 and Pioneer P3-DX robot in real-world as Σ_3.

MFRL differs from transfer learning (Taylor, Stone, and Liu 2007), where a transfer of parameters is allowed only in one direction. The MFRL algorithm starts in Σ_1. Once it learns a sufficiently good policy in Σ_1, it switches to a higher fidelity simulator. If it observes that the policy learned in the lower fidelity simulator is no longer optimal in the higher fidelity simulator, it switches back to the lower fidelity simulator. (Cutler, Walsh, and How 2015) showed that the resulting algorithm has polynomial sample complexity and minimizes the number of samples required for the highest fidelity simulator, i.e., the real world, under some technical conditions.

The original MFRL algorithm uses the Knows-What-It-Knows framework (Li et al. 2011) to learn the transition and reward functions at each level. The reward and transition for each state-action pair are learned independently of others. While this is reasonable for general agents, when planning for physically-grounded robots, we can exploit the spatial correlation between neighboring state-action pairs to speed up the learning.

We use Gaussian Process (GP) regression (Rasmussen and Williams 2006) as a function approximator to speed up learning in the MFRL framework. GPs can predict the learned function value for any query point, and not just for a discretized state-action pair. Furthermore, GPs can exploit the correlation between nearby state-action values by an appropriate choice of a kernel. GPs have been used in conjunction with policy search methods to obtain optimal policies in simulation-aided reinforcement learning (Cutler and How 2016; M. Cutler and J. P. How 2015). We take this further by using GPs in the MFRL setting.

In MFRL, the state-space of Σ_i is a subset of the state space of Σ_j for all $j > i$. Therefore, when the MFRL algorithm switches from Σ_i to Σ_{i+1} it already has a policy (better than naive) for states in $\Sigma_{i+1} \setminus \Sigma_i$. Thus, GPs are particularly suited for MFRL, which we verify through our simulation results.

Our main contribution in this paper is to leverage GP regression for model-free MFRL by directly estimating the optimal Q-values (GPQ-MFRL) and subsequently calculating the optimal policy. We verify the performance of the GP-based MFRL algorithm through simulations as well as experiments with a ground robot. Our empirical evaluation shows that the GP-based MFRL algorithm learns the optimal policy faster than the original MFRL algorithm using even fewer real-world samples. The original MFRL work was for a model-based approach. We extend it to introduce a model-free version. Model-free algorithms have been used in conjunction with transfer learning in the past (Taylor, Stone, and Liu 2007).

The rest of the paper is organized as follows. In the next section, we present the background on RL and GPs followed by a survey of related work. Next, we present the GP-MFRL algorithm (GPQ-MFRL) followed by experimental results along with comparisons with other MFRL and non-MFRL techniques. We conclude with a discussion of the future work.

Background

Reinforcement Learning

RL problems can be formulated as a Markov Decision Process (MDP): $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $\mathcal{P}(s_t, a_t, s_{t+1}) \mapsto [0,1]$, reward function $\mathcal{R}(s_t, a_t) \mapsto \mathbb{R}$ and discount factor $\gamma \in [0,1)$ (Sutton and Barto 1998; Puterman 2014). A policy $\pi : \mathcal{S} \to \mathcal{A}$ maps states to actions. Together with the initial state s_0, a policy forms a trajectory $\zeta = \{[s_0, a_0, r_0], [s_1, a_1, r_1], \ldots\}$ where $a_t = \pi(s_t)$. r_t and s_{t+1} are sampled from the reward and transition functions, respectively.

We consider a scenario where the goal is to maximize the infinite horizon discounted reward starting from a state s_0. The value function for a state s_0 is defined as $\mathcal{V}^\pi(s_0) = \mathbb{E}[\sum_{t=0}^{t=\infty} \gamma^t r_t(s_t, a_t) | a_t = \pi(s_t)]$. The state-action value function or Q-value of each state-action pair under policy π is defined as $Q^\pi(s,a) = \mathbb{E}[\sum_{t=0}^{t=\infty} \gamma^t r_{t+1}(s_{t+1}, a_{t+1}) | s_0 = s, a_0 = a]$ which is the expected sum of discounted rewards obtained starting from state s, taking action a and following π thereafter. The optimal Q-value function Q^* for a state-action pair (s,a) satisfies $Q^*(s,a) = \max_\pi Q^\pi(s,a) = \mathcal{V}^*(s)$ and can be written recursively as,

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r(s_t, a_t) + \gamma \mathcal{V}^*(s_{t+1})]. \quad (1)$$

Our objective is to find the optimal policy $\pi^*(s) = \text{argmax}_a Q^*(s,a)$ when $\mathcal{R}$ and $\mathcal{P}$ are not known to the agent. In model-based approaches, the agent learns $\mathcal{R}$ and $\mathcal{P}$ first and then finds an optimal policy by calculating optimal Q-values from Equation 1. The most commonly used model-based approach is VI (Jung and Stone 2010; Brafman and Tennenholtz 2002). We can also directly estimate the optimal Q-values (model-free approaches) (Strehl et al. 2006; Grande, Walsh, and How 2014) or directly calculate the optimal policy (policy-gradient approaches) (Sutton et al. 2000). The most commonly used model-free algorithm is Q-learning (Watkins and Dayan 1992). For our GPQ-MFRL implementation, we use Q-learning to perform the policy update using GP regression.

In this work, we focus on the model-free version of GP based MFRL since they are computationally and memory efficient (Brafman and Tennenholtz 2002).

Gaussian Processes

GPs are Bayesian non-parametric function approximators. GPs can be defined as a collection of infinitely many random variables, any finite subset $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ [1] of which is jointly Gaussian with mean vector $\mathbf{m} \in \mathbb{R}^k$ and covariance matrix $\mathbf{K} \in \mathbb{R}^{k \times k}$ (Rasmussen and Williams 2006).

Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ denote the set of the training inputs. Let $\mathbf{y} = \{y_1, \ldots, y_k\}$ denote the corresponding training outputs. GPs can be used to predict the output value at a new test point, $\mathbf{x}$, conditioned on the training data. Predicted output value at $\mathbf{x}$ is normally distributed with mean $\hat{\mu}(\mathbf{x})$ and variance $\hat{\sigma}^2(\mathbf{x})$ given by,

$$\hat{\mu}(\mathbf{x}) = \mu(\mathbf{x}) + \mathbf{k}(\mathbf{x}, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \omega^2 \mathbf{I}]^{-1} \mathbf{y}, \quad (2)$$

$$\hat{\sigma}^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x}, \mathbf{X})[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \omega^2 \mathbf{I}]^{-1} \mathbf{k}(\mathbf{X}, \mathbf{x}) + \omega^2, \quad (3)$$

where $\mathbf{K}(\mathbf{X}, \mathbf{X})$ is the kernel. The entry $\mathbf{K}_{\mathbf{x}_l, \mathbf{x}_m}$ gives the covariance between two inputs $\mathbf{x}_l$ and $\mathbf{x}_m$. $\mu(\mathbf{x})$ in Equation 2 is the prior mean of output value at $\mathbf{x}$.

We use a zero-mean prior and a squared-exponential kernel where $\mathbf{K}_{\mathbf{x}_l, \mathbf{x}_m}$ is given by,

$$\mathbf{K}_{\mathbf{x}_l, \mathbf{x}_m} = \sigma^2 \exp\left(-\frac{1}{2} \sum_{d=1}^{d=D} \left(\frac{\mathbf{x}_{dl} - \mathbf{x}_{dm}}{l_d} \right)^2 \right) + \omega^2, \quad (4)$$

σ^2, l_d and ω^2 are hyperparameters that can be either set by the user or learned online through the training data.

In the GPQ-MFRL algorithm, we use GPs to learn Q-values. GPs are proved to be consistent function approximators in RL with convergence guarantees (Melo, Meyn, and Ribeiro 2008; Chowdhary et al. 2014). A set of state-action pairs is the input to GP and Q–values are the output/observation values to be predicted.

[1]Upper and lower bold face letters represent matrices and vectors respectively. Scalar values are represented by a lower-case letter.

Related Work

In model-based approaches, GPs are commonly used to learn transition models for agents moving in the real world (Dames, Tokekar, and Kumar 2015) and have been used in RL to learn the transition function (Rasmussen and Kuss 2003) and the reward function (Deisenroth 2010). GPs have also been used in model-free approaches for approximating the Q-values in continuous state-action spaces (Engel, Mannor, and Meir 2003). This was extended to the GP-SARSA algorithm which includes online action selection and policy improvement steps (Engel, Yaakov and Mannor, Shie and Meir, Ron 2005). The authors used the posterior variance to compute confidence intervals around the value estimate and note that the variance could be used for exploration.

Using multiple approximations of real-world environments has previously been considered in the literature (Abbeel, Quigley, and Ng 2006; Taylor, Stone, and Liu 2007; Torrey and Shavlik 2009). (Yao and Doretto 2010) extended the transfer learning framework for transferring knowledge from multiple sources. (Yosinski et al. 2014) show the transfer of features in deep neural networks and demonstrate that initializing a network with transferred features from almost any number of layers can generalize to fine-tuning to the target dataset. Unlike these methods, the MFRL algorithm allows for bi-directional switching, where the agent is allowed to go back to lower fidelity simulator to gather additional samples.

The MFRL algorithm was introduced by (Cutler, Walsh, and How 2015) where they showed how to leverage the model-based RMax algorithm to reduce the number of samples. Our empirical results demonstrate that the number of samples for MFRL can be brought further down by leveraging GPs.

Algorithm Description

In this section, we first describe our algorithm. We compare the proposed algorithm with baseline strategies through simulations. A flowchart of the proposed algorithm is shown in Figure 2.

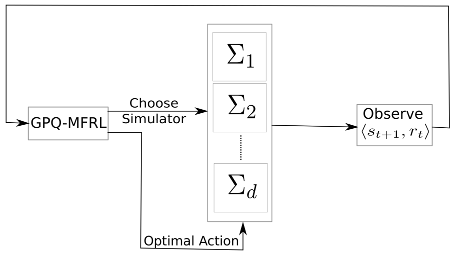


Figure 2: Overview of our model-free algorithm (GPQ-MFRL). Simulators are represented by $\Sigma_1, \Sigma_2, \ldots, \Sigma_d$. GPQ-MFRL directly estimates Q-values using GPs.

GPQ-MFRL Algorithm

The agent learns the optimal Q-values using GPs directly, instead of learning the model first. The underlying assumption is that nearby state-action pairs will produce similar Q-values. This assumption can also be applied to problems where the states and actions are discrete but the transition function implies some sense of continuity. We choose the squared-exponential kernel because it models the spatial correlation we expect to see in a robot. However, any appropriate kernel can be used. We use a separate GP per simulator to estimate the Q-values using only data collected in that simulator.

Algorithm 1 GPQ-MFRL Algorithm

```
1: procedure
2: Input: confidence parameters $\sigma_{th}$ and $\sigma_{th}^{sum}$; simulator chain $\langle \boldsymbol{\Sigma},$ fidelity parameters $\boldsymbol{\beta},$ state mappings $\boldsymbol{\rho} \rangle$; $\mathcal{L}$.
3: Initialize: $\hat{Q}_i$ = initialize GP for $i \in \{1, \ldots, d\}$; state $s_0$ in simulator $\Sigma_1$; $i \leftarrow 1$; $change$ = FALSE.
4: Initialize: $t \leftarrow 0$; $\mathcal{D}_i \leftarrow \{\}$ for $i \in \{1, \ldots, d\}$.
5: while terminal condition is not met
6:     $a_t \leftarrow$ CHOOSEACTION$(s_t, i)$
7:     if $\sigma_i(s_t, a_t) \leq \sigma_{th}$: $change$ = TRUE
8:     if $\sigma(\rho_i(s_t), a_t) > \sigma_{th}$ and $change$ and $i > 1$
9:         $s_t \leftarrow \rho_i(s_t),\ i \leftarrow i - 1$, continue
10:    $\langle r_t, s_{t+1} \rangle \leftarrow$ execute action $a_t$ in $\Sigma_i$
11:    append $\langle s_t, a_t, s_{t+1}, r_t \rangle$ to $\mathcal{D}_i$
12:    $\mathcal{Y}_i \leftarrow \{\}$
13:    for $\langle s_t, a_t, s_{t+1}, r_t \rangle \in \mathcal{D}_i$ //batch training//
14:        $y_t \leftarrow r_t + \gamma \max_a \hat{Q}_i(s_{t+1}, a)$
15:        append $\langle s_t, a_t, y_t \rangle$ to $\mathcal{Y}_i$
16:    $\hat{Q}_i \leftarrow$ update GP$_i$ using $\mathcal{Y}_i$
17:    if $\sum_{j=t}^{j=t-\mathcal{L}} \sigma_i(s_j, a_j) \leq \sigma_{th}^{sum}$ and $t > \mathcal{L}$ and $i < d$
18:        $s_t \leftarrow \rho_{i+1}^{-1}(s_t),\ i \leftarrow i + 1$
19:        $change$ = FALSE
20: end procedure
21:
22: procedure CHOOSEACTION$(s, i)$
23: for $a \in \mathcal{A}(s)$
24:     $Q(s, a) = \hat{Q}_{i-1}(\rho_i(s), a) + \beta_i$
25:     for $k \in \{i, \ldots, d\}$
26:         $s_k = \rho_k^{-1} \ldots \rho_{i+2}^{-1} \rho_{i+1}^{-1}(s)$
27:         if $\sigma_k(s_k, a) \leq \sigma^{th}$: $Q(s, a) = \hat{Q}_k(s_k, a)$
28: return $\arg\max_a Q(s, a)$
29: end procedure
```

Algorithm 1 gives the details of the proposed framework. GPQ-MFRL continues to collect samples in the same simulator until the agent is confident about its optimal actions. If the running sum of the variances is below a threshold (Line 17), this suggests that the robot has found a good policy with high confidence in the current simulator and it must advance to the next one (Line 18).

GPQ-MFRL keeps track of the variance of the $\mathcal{L}$ most recently visited state-action pairs in the current simulator. If the running sum of the variances is below a threshold (Line 15), this suggests that the robot is confident about its actions in the current simulator and can advance to the next one. In the original work (Cutler, Walsh, and How 2015), the agent switches to the higher fidelity simulator after a certain number of known state-action pairs were encountered. In our implementation (Line 7), the model of current environment changes if the posterior variance for a state-action pair drops below a threshold value (*i.e.*, agent has a sufficiently accurate estimate of the transitions from that state). The algorithm checks if the agent has a sufficiently accurate estimate of optimal Q-values in the previous simulator (Line 8). Lines 10–15 describe the main body of the algorithm where the agent records the observed transitions in $\mathcal{D}_i$. We update target values (Line 14) for every transition as more data gets collected in $\mathcal{D}_i$ (Line 13). The GP model is updated after every step (Line 16).

The agent utilizes the experiences collected in higher simulators (Lines 25–27) to choose the optimal action in the current simulator (Line 6). Specifically, it checks for the maximum fidelity simulator in which the posterior variance for (s, a) is less than a threshold σ_{th}. If one exists, it utilizes the Q-values from the highest known simulator to choose the next action in the current simulator. If no such higher simulator exists, the Q-values from the previous simulator (Line 24) are considered to choose the next action in the current simulator with an additive fidelity parameter β.

GPQ-MFRL performs a batch retraining every time the robot collects the new sample in a simulator (Lines 13–15). During the batch retraining, the algorithm updates the target values in previously collected training data using the knowledge gained by collecting new samples. Then these updated target values are used to predict the Q-values using GPs (Line 16). As the amount of data grows, updating the GP can become computationally expensive. However, we can prune the dataset using sparse GP techniques (Engel, Yaakov and Mannor, Shie and Meir, Ron 2005). It is non-trivial to choose values for confidence bounds but for the current experiments we chose the σ_{th}^{sum} to be ten percent of the maximum Q-value possible and σ_{th} to be one fifth of σ_{th}^{sum}.

Results

We use three environments to simulate GPQ-MFRL. Σ_1 is Python-based simulator Pygame (Shinners 2011), Σ_2 is a Gazebo environment, and Σ_3 is the real world.

Evaluating the GPQ-MFRL Algorithm

We use three environments (Figure 1) to demonstrate the GPQ-MFRL algorithm. The task of the robot is to navigate through a given environment without crashing into the obstacles, assuming the robot has no prior information about the environments. There is no goal state.

The robot has a laser sensor that gives the distance readings from the obstacles along seven equally spaced directions. The angle between two consecutive measurement directions is $\frac{\pi}{8}$ radians and range of measurements is 5 meters. The actual robot has a Hokuyo laser sensor that operates in the same configuration. Distance measurements along the seven directions serve as the state in the environment. Hence, we have a seven-dimensional continuous state space: $\mathcal{S} \in (0, 5]^7$ (Figure 1), where individual state dimension corresponds to the distance measurement along a particular direction. Note that this is a different state representation than the standard one used in navigation where the X and Y coordinates of the robot are used as the state. Since we use the sensor input directly as the state, the learned policy maps the sensor inputs directly to the actions, thereby learning a polygon that avoids collisions. This can easily generalize to any environment.

The linear speed of the robot is held constant at 0.2 m/sec. The robot can choose its angular velocity from nineteen possible options: $\{-\frac{\pi}{9}, -\frac{\pi}{8}, \ldots, \frac{\pi}{9}\}$. The reward in each state is set to be the sum of laser readings from seven directions except when the robot hits the obstacle when it gets a reward of -50.

We train the GP regression, $Q(\mathbf{s}, a) : \mathbb{R}^8 \to \mathbb{R}$. Hyperparameters of the squared-exponential kernel were calculated off-line by minimizing the negative log marginal likelihood of 2000 training points which were collected by letting the robot run in the real world directly. The parameter values for experiments in this section are given in Table 1.

Table 1: Parameters used in GPQ-MFRL

Description	Type	Value
Hyperparameters	σ	102.74
	l	[2.1, 5.1, 14, 6.2, 15, 2, 2, 1]
	ω^2	20
Confidence parameters	σ_{th}^{sum}	60
	σ_{th}	15
Algorithm	$\mathcal{L}$	5

Average Cumulative Reward in the Real-World In Figure 3, we compare GPQ-MFRL algorithm with three other baseline strategies by plotting the average cumulative reward collected by the robot as a function of samples collected in the real world. Three baseline strategies are,

1. Directly collecting samples in the real world without the simulators (Direct),
2. Collect hundred samples in one simulator and transfer the policy to the Pioneer robot with no further learning in the real world (Frozen Policy) and
3. Collect hundred samples in one simulator and transfer the policy to the robot while continuing to learn in the real world (Transferred Policy).

Policy Improvement over Time Figure 4 shows the absolute percentage change in the sum of the value functions with respect to last estimated sum of value functions and average predictive variance for states $\{1, 3, 5\}^7$ in all the three simulators. Observe that initially most of the samples are collected in the simulator, whereas over time the samples are collected mostly in the real world. The simulators help the robot to make its value estimates converge quickly as observed by a sharp dip in the first white region. Note that GP updates for i^{th} simulator ($\hat{Q}_i$) are made only when the robot is running in i^{th} simulator.

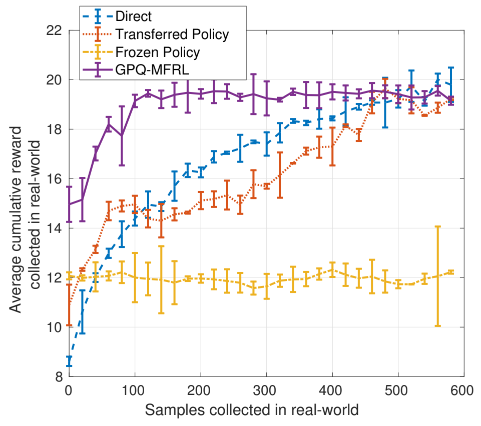
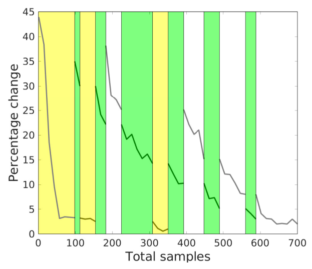
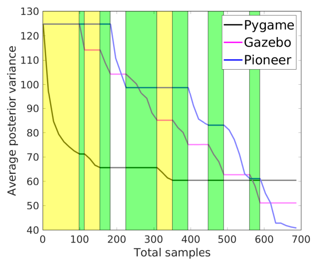


Figure 3: Average cumulative reward collected by Pioneer in real-world as a function of the samples collected in real-world. The plot shows the average and standard deviation of 2 trials.



(a) Sum of absolute change in value functions



(b) Average variances in value function estimations

Figure 4: Yellow, green and white regions correspond to the samples collected in the Pygame, Gazebo and real-world environments respectively. Plots are for state set $\{1, 3, 5\}^7$.

Discussion and Future Work

The GP-based MFRL algorithm provides a general RL technique that is particularly suited for robotics. We demonstrated empirically that the GP-based MFRL algorithm finds the optimal policies using fewer samples than the baseline algorithms. We plan to analyze the algorithms in order to provide theoretical bounds for the sample complexity. (Strehl et al. 2006) show that the sample complexity of RMax algorithm after ignoring logarithmic factors is $\tilde{O}\left(\frac{S^2A}{\epsilon^3(1-\gamma)^6}\right)$. Here S is the size of state space, A is the number of actions available to the agent, γ is the discount factor and ϵ denotes the desired accuracy until the algorithm converges.

One disadvantage of using GPs is that as the number of observations increase, the time taken to perform GP updates also increases in with cubic complexity. However, we can use adaptive sample selection techniques (Osborne 2010) as well as numerical optimization techniques (Engel, Yaakov and Mannor, Shie and Meir, Ron 2005) to speed up this process.

References

Abbeel, P.; Quigley, M.; and Ng, A. Y. 2006. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, 1–8. ACM.

Brafman, R. I., and Tennenholtz, M. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3(Oct):213–231.

Chowdhary, G.; Liu, M.; Grande, R.; Walsh, T.; How, J.; and Carin, L. 2014. Off-policy reinforcement learning with Gaussian processes. *IEEE/CAA Journal of Automatica Sinica* 1(3):227–238.

Cutler, M., and How, J. P. 2016. Autonomous drifting using simulation-aided reinforcement learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 5442–5448.

Cutler, M.; Walsh, T. J.; and How, J. P. 2015. Real-world reinforcement learning via multifidelity simulators. *IEEE Transactions on Robotics* 31(3):655–671.

Dames, P.; Tokekar, P.; and Kumar, V. 2015. Detecting, localizing, and tracking an unknown number of moving targets using a team of mobile robots. In *International Symposium on Robotics Research (ISRR)*.

Deisenroth, M. P. 2010. *Efficient reinforcement learning using Gaussian processes*, volume 9. KIT Scientific Publishing.

Engel, Y.; Mannor, S.; and Meir, R. 2003. Bayes meets bellman: The gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 154–161.

Engel, Yaakov and Mannor, Shie and Meir, Ron. 2005. Reinforcement learning with Gaussian Processes. In *Proceedings of the 22nd international conference on Machine learning*, 201–208. ACM.

Grande, R.; Walsh, T.; and How, J. 2014. Sample efficient reinforcement learning with gaussian processes. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 1332–1340.

Jung, T., and Stone, P. 2010. Gaussian processes for sample efficient reinforcement learning with rmax-like exploration. In *Proceedings of the European Conference on Machine Learning*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436–444.

Li, L.; Littman, M. L.; Walsh, T. J.; and Strehl, A. L. 2011. Knows what it knows: a framework for self-aware learning. *Machine learning* 82(3):399–443.

M. Cutler and J. P. How. 2015. Efficient reinforcement learning for robots using informative simulated priors. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2605–2612.

Melo, F. S.; Meyn, S. P.; and Ribeiro, M. I. 2008. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th international conference on Machine learning*, 664–671. ACM.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Osborne, M. 2010. *Bayesian Gaussian processes for sequential prediction, optimisation and quadrature*. Ph.D. Dissertation, University of Oxford.

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Rasmussen, C. E., and Kuss, M. 2003. Gaussian processes in reinforcement learning. NIPS.

Rasmussen, C. E., and Williams, C. K. I. 2006. *Gaussian Processes for Machine Learning*. the MIT Press.

Shinners, P. 2011. Pygame. `http://pygame.org/`.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Strehl, A. L.; Li, L.; Wiewiora, E.; Langford, J.; and Littman, M. L. 2006. Pac model-free reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, 881–888. ACM.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

Taylor, M. E.; Stone, P.; and Liu, Y. 2007. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research* 8(Sep):2125–2167.

Torrey, L., and Shavlik, J. 2009. Transfer learning: Handbook of research on machine learning applications and trends: Algorithms, methods, and techniques.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

Yao, Y., and Doretto, G. 2010. Boosting for transfer learning with multiple sources. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, 1855–1862. IEEE.

Yosinski, J.; Clune, J.; Bengio, Y.; and Lipson, H. 2014. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, 3320–3328.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Towards Perception Aware Task-Motion Planning

Antony Thomas, Sunny Amatya, Fulvio Mastrogiovanni, Marco Baglietto
Department of Informatics, Bioengineering, Robotics, and Systems Engineering
University of Genoa, Via All'Opera Pia, 13
{antony.thomas@dibris.unige.it, sunnyamatya@gmail.com, fulvio.mastrogiovanni@unige.it, marco.baglietto@unige.it}

Abstract

We present an integrated Task-Motion Planning (TMP) framework in belief space. Autonomous robots operating in real world complex scenarios require planning in the discrete (task) space and the continuous (motion) space. We develop a framework for integrating belief space reasoning within a hybrid task planner. The expressive power of PDDL+ combined with heuristic driven semantic attachments performs the propagated and posterior belief estimates while planning. The underlying methodology for the development of the hybrid planner is discussed, providing suggestions for further improvements and future work. Furthermore, our approach is unified with ROSPlan and we validate key aspects of our approach using a realistic synthetic simulation.

1 Introduction

Autonomous robots operating in complex real world scenarios require different levels of planning to execute their tasks. High-level (task) planning helps to break down a given set of tasks into a sequence of sub-tasks, depending on the required level of abstraction. Actual execution of each of these subtasks would require low-level control actions (i.e., motions). Hence, planning should be performed in the task-motion or the discrete-continuous space.

Traditionally, task planning and motion planning have evolved as two independent fields. Since the seminal work of Fikes and Nilsson (Fikes and Nilsson 1971), planning has emerged as a specific field within AI. Techniques like, planning graphs, planning as satisfiability and heuristic-search planning have led to dominant task planning approaches like Fast Downward (Helmert 2006) and FF (Hoffmann and Nebel 2001) planning systems, that are capable of highly efficient planning in larger domains, handling mathematical expressions and temporal aspects. Similarly, the field of motion planning has evolved profusely. Probabilistic Roadmaps (PRMs) (Kavraki et al. 1996; Kavraki, Kolountzakis, and Latombe 1998) and Rapidly-exploring Random Trees (RRTs) (Kuffner and LaValle 2000) are the two notable approaches for high-dimensional motion planning. These are *Sampling based approaches* providing faster execution and better efficiency.

In recent years, combining high-level task planning with the low-level motion planning has been a subject of great interest among the Robotics and Artificial Intelligence (AI) community. This is inevitable as one of the ultimate goals in Robotics is to create autonomous agents accepting high-level task descriptions and executing them without further human intervention. Planning frameworks such as the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) mainly focus on high-level task planning supposing that the geometric preconditions (e.g., grasping poses for a pick-up task (Srivastava et al. 2014)) for the robot motion to carry out these tasks are achievable. However, in reality, such an assumption can be catastrophic as an action or sequence of actions generated by the task planning algorithm might turn out to be unfeasible at the controller execution level.

Let us consider a simple scenario where a robot is given the task of picking up an object. In terms of task planning, a *pick_up* action would suffice, subject to satisfying the action preconditions, i.e., the robot hand being free and the object being graspable. However, it is possible that the robot is too close to the object and the *pick_up* action cannot be performed due to robot's end-effector's reachability workspace. This would require the robot to assume a different grasping pose, invoking a motion command that leads to a suitable position. Though a simple scenario, it clearly illustrates the need for a combined TMP strategy. Several recent works (Cambon, Alami, and Gravot 2009; Kaelbling and Lozano-Pérez 2012; Dornhege et al. 2012; Srivastava et al. 2014; Toussaint 2015; Dantam et al. 2018) have motivated the need for a combined Task-Motion Planning (TMP) approach pointing out the drawbacks of treating them as separate.

Given an initial state and a suitable goal state, Task-Motion Planning (TMP) synthesizes a plan interleaving discrete high-level tasks with continuous low-level motion. In robotics planning, reaching a goal pose x_g from an initial pose x_0 requires continuous collision-free motion planning. However, reaching a goal alone is not sufficient as most often we require the goal condition to be satisfied subject to application dependent costs. As a result, certain decisions are to be made with regards to visiting specific landmarks, the order and type of actions to be performed. Yet, the most important challenge for TMP problems is finding the right correspondence between the task planner and the motion planner. Given a discrete action, a TMP should be able to recognize the corresponding geometry requirements to trigger the action. Similarly, once the action is triggered, the corresponding motion planning problems are to be identified.

Furthermore, real-world scenarios often induce uncertainties. Such uncertainties arise due to insufficient knowledge about the environment, inexact robot motion or imperfect sensing. In such scenarios, the robot poses or other variables of interest can only be dealt with in terms of probabilities. Planning is therefore done in the *belief* space, which corresponds to the probability distributions over possible robot states. Consequently, for efficient planning and decision making, it is required to reason about future belief distributions due to candidate actions and the corresponding expected observations. Such a problem falls under the category of partially observable Markov decision processes (POMDPs). Hence, the task planner should be capable of reasoning in the belief space while synthesizing a plan. Besides, the task planner also requires some amount of geometrical information about the environment and the robot itself, the lack of which leads to undesirable plans. At the execution level, the motion planner might encounter unexpected scenarios notwithstanding the plan provided. This calls for a re-plan, updating the task planner with the new belief, resulting in a cyclic interdependency. Consequently, both task and motion planning are interdependent and should not be considered as separate processes.

Our contributions in this paper are as follows: (1) developing an integrated TMP algorithm for planning in the belief space. (2) The expressive power of PDDL+ is exploited to simulate robot motion and the belief updates within PDDL+. Our domain description can hence be employed for any mobile robot planning problem in general. (3) We sample landmark observable viewpoints, thereby facilitating perception aware TMP. Our sampling strategy is also directed towards parsimoniously selecting connected waypoints thereby reducing the state space explosion (4) The developed framework is also unified with ROSPlan (Cashmore et al. 2015), which is a framework for embedding task planning in ROS and hence can be easily adapted by the planning community. Furthermore, we provide an interface for PDDL+ planning in ROSPlan.

2 Related Work

The genesis of TMP can be credited to Fikes and Nilsson for their work on STRIPS (Fikes and Nilsson 1971) which further led to the Shakey project (Nilsson 1984). Shakey's planner had access to basic geometric knowledge like the objects in a room, connectivity between rooms. However, it performed a logical search first, assuming that the resulting robot motion plans can be formulated. This assumption limits the capability of the agent as the high-level actions may turn out to be non executable due to geometric limitations. Later works either carried out the generated plans, validating them using a robot motion planner (Dornhege et al. 2009) or performed a combined search in the logical and geometric spaces using a state composed of the both the symbolic and geometric paths (Cambon, Alami, and Gravot 2009). The aSyMov planner used in (Cambon, Alami, and Gravot 2009) used a combination of Metric-FF (Hoffmann 2003) and a sampling based motion planner. In contrast, we use a hybrid temporal task planner (Piotrowski et al. 2016) incorporating robot state uncertainty. Srivastava et al. implicitly incorporate geometric variables in a PDDL-based planning model (Srivastava et al. 2014). An interface layer then converts PDDL plans to numeric values of the symbols to check the validity of each action in the configuration space.

Many of the planning problems need to plan well ahead of time which also results in increased dimensionality, as more and more objects and constraints get added. Longer planning horizons and higher dimensionality directly affects the computational time for these planners. Kaelbling and Lozano-Péres (2012) propose a hierarchical approach that tightly integrates the logical and geometric planning. The complexities arising out of long horizon planning are tackled to the extent that planning is done at different levels of abstraction, thereby reducing the long horizons to a number of feasible sub-plans of shorter horizon. This regression-based planner assumes that the actions are reversible while backtracking. In contrast to their earlier work the serializability assumption of the subgoals is relaxed. Kaelbling and Lozano-Péres (2013) further extended their work to consider the current state uncertainty, modeling the planning problem in the belief space. Uncertain outcomes are modeled by converting a Markov decision processes (MDP) into a weighted graph, thereby modifying their earlier approach of *hierarchical planning in the now*. Belief update is then performed when observations are obtained. Phiquepal and Toussaint (2017) discuss an ongoing work for TMP under partial observability, computing long-horizon policies that are arborescent in nature.

The above discussed approaches focus on finding feasible plans sacrificing optimality and hence emphasizes on performance. Toussaint (2015) performs optimization over an objective function based on the final geometric configuration (and the cost thereby), finding approximately locally optimal solutions by minimizing the objective function. The planning problem is modeled as a constraint satisfaction problem with symbolic states used to define the constraints in the optimization. Lozano-Péres and Kaelbling (2014) model the motion planning as a constraint satisfaction problem over a subset of the configuration space. Iteratively Deepened Task and Motion Planning (IDTMP) is a constraint based task planning approach that incorporates geometric information (motion feasibility) at the task planning level (Dantam et al. 2018). In our approach, the waypoints fed into the task planner are generated using the motion planner, similar to the motion planner information that guides the IDTMP task planner.

3 Preliminaries

Let x_k denote the robot pose at any time k defined by $x_k \doteq (x, y, \theta)$, x and y are the robot Cartesian coordinates and θ is the heading. We use z_k to denote the measurement acquired at time k and u_k for the control action applied at time k. Extended Kalman Filter (EKF) is used to represent the robot belief at a given time, using the robot state mean μ_k and covariance Σ_k. The robot dynamics is modeled using the standard odometry based motion model

$$\begin{aligned} x' &= x + \delta_{trans} \cdot \cos(\theta + \delta_{rot1}) \\ y' &= y + \delta_{trans} \cdot \sin(\theta + \delta_{rot1}) \\ \theta' &= \theta + \delta_{rot1} + \delta_{rot2} \end{aligned} \tag{1}$$

where $u_k \doteq (\delta_{rot1}, \delta_{trans}, \delta_{rot2})$ is the control applied. For brevity we write Eq. 1 as $x_{k+1} = f(x_k, u_k)$. We assume Gaussian noise and the process covariance is given by

$$W_k = \begin{bmatrix} \alpha_1 \cdot \delta^2_{rot1} + \alpha_2 \cdot \delta^2_{trans} & 0 & 0 \\ 0 & \alpha_3 \cdot \delta^2_{trans} + \alpha_4 \cdot (\delta^2_{rot1} + \delta^2_{rot2}) & 0 \\ 0 & 0 & \alpha_2 \cdot \delta^2_{trans} + \alpha_1 \cdot \delta^2_{rot2} \end{bmatrix} \tag{2}$$

where α_1-α_4 are robot-specific error parameters (Thrun, Burgard, and Fox 2005) modeling the accuracy of the robot motion. To process the landmarks in the environment we measure the range and the bearing of the landmark relative to the robot's local coordinate frame. It is to be noted that we assume the data association problem is solved and hence given a measurement we know the corresponding landmark that generated it. Such a model can be represented by

$$z_k = \begin{bmatrix} r \\ \phi \end{bmatrix} + v_k \ , \ v_k \sim \mathcal{N}(0, Q_k) \tag{3}$$

where r, ϕ is the range and bearing respectively and v_k the zero-mean Gaussian noise. For brevity, Eq. 3 will be written as $z_k = h(x_k, l^i)$

The motion (Eq. 1) and observation (Eq. 3) models can be written probabilistically as $p(x_{k+1}|x_k, u_k)$ and $p(z_k|x_k)$ respectively. Given an initial distribution $p(x_0)$, and the motion and observation models, the posterior probability distribution at time k can be written as

$$p(X_k|Z_k, U_{k-1}) = p(x_0) \prod_{i=1}^{k} p(x_k|x_{k-1}, u_{k-1}) p(z_k|x_k) \tag{4}$$

where $X_k = \{x_0, ..., x_k\}$, $Z_k = \{z_0, ..., z_k\}$ and $U_k = \{u_0, ..., u_{k-1}\}$. This posterior probability distribution is the *belief* at time k, denoted by $b[X_k] \sim \mathcal{N}(\mu_k, \Sigma_k)$. Similarly, given an action u_k, the propagated belief can be written as

$$b[\bar{X_{k+1}}] = p(X_k|Z_k, U_{k-1}) p(x_{k+1}|x_k, u_k) \tag{5}$$

Given the current belief $b[X_k]$, the control u_k, the propagated belief parameters can be computed using the standard EKF prediction as

$$\begin{aligned} \bar{\mu}_{k+1} &= f(\mu_k, u_k) \\ \bar{\Sigma}_{k+1} &= F_k \Sigma_k F_k^T + V_k W_k V_k^T \end{aligned} \tag{6}$$

where F_k and V_k are the Jacobians of $f(\cdot)$ with respect to x_k and u_k respectively. For ease of representation, we denote $R_k \doteq V_k W_k V_k^T$. Upon receiving a measurement z_k, the posterior belief $b[X_{k+1}]$ is computed using the EKF update equations

$$\begin{aligned} K_k &= \bar{\Sigma}_{k+1} H_k^T (H_k \bar{\Sigma}_{k+1} H_k^T + Q_k)^{-1} \\ \mu_{k+1} &= \bar{\mu}_{k+1} + K_k (z_{k+1} - h(\bar{\mu}_{k+1}, l^i)) \\ \Sigma_{k+1} &= (I - K_k H_k) \bar{\Sigma}_{k+1} \end{aligned} \tag{7}$$

where H_k is the Jacobian of $h(\cdot)$ with respect to x, K_k is the Kalman gain and $\mathbb{I} \in \mathbb{R}^{3 \times 3}$.

4 TMP Design and Implementation

In this Section we detail our TMP planner concept and approach. We begin by making the following observation. Planning in the belief space to obtain an optimal control policy essentially requires synthesizing a sequence of actions that minimize an application dependent objective function. Finding such an action sequence inherently involves searching in the motion space. Consequently, we employ task planning to perform this search.

4.1 Rationale and Scenario

PDDL based planning frameworks are restricted, as they are incapable of handling rigorous numerical calculations. Most approaches perform such calculations via an external module or *semantic attachments*, e.g. (Dornhege et al. 2012). The term semantic attachment was coined by Weyhrauch (1980) to describe attaching algorithms to function and predicate symbols via external procedure. Yet, the effects returned by these semantic attachments are not exploited in identifying *helpful actions* and hence do not provide any heuristic guidance, deeming the task unsolvable most often. An action is considered *helpful* if it achieves at least one of the lowest level goals in the relaxed plan to the state at hand (Hoffmann 2003). Recently Bernardini et al. (2017) developed a PDDL based POPF-TIF planner to implicitly trigger such external calls via a specialized semantic attachments called *external advisors*. They classify variables into direct, indirect and free variables. Direct (free) variables are the normal PDDL function variables whose values are changed in the action effects, in accordance with the PDDL semantics. The indirect variables are affected by the changes in the direct variables. A change in a direct variable triggers the external advisor which in turn updates the indirect variables. POPF-TIF is based on the temporal extension of the metric-FF planner (Hoffmann 2003). An intriguing feature of the planner is that it uses approximate values of the indirect variables at the Temporal Relaxed Plan Graph (TRPG) construction stage resulting in an efficient goal-directed search. During the forward state space search, the external advisor is called, updating the indirect variables with the exact values.

Leveraging the ROSPlan framework (Cashmore et al. 2015) and using semantic attachments that incorporate heuristic evaluation during the Relaxed Plan Graph (RPG) construction, we develop a hybrid planning framework capable of reasoning in the robot belief space while synthesizing a plan. We use PDDL+ (Fox and Long 2006) to model the planning task, providing the robot with a sequence of actions that can be passed on to the low-level controller for execution. PDDL+ provides the ability to model continuous temporal change via *processes* and discrete exogenous activities in the environment via *events*. The processes are similar to durative actions and the events are akin to instantaneous actions. However, processes and events are distinct from actions since a process or an event is triggered as soon as its precondition is satisfied whereas an action trigger depends on the planner search strategy. State uncertainty is incorporated in our model and synthesizing an efficient plan requires performing the belief updates within the task planner. PDDL+ *processes* enable the simulation of robot motion with time and the *events* are leveraged to perform the corresponding belief updates. In our case, we use the DiNo planner (Piotrowski et al. 2016) since it enables heuristic search for linear and non-linear systems using the entire set of PDDL+ features.

In this paper, we consider a mobile robot in a known environment (i.e., map is given) with uncertainty in its initial pose. The set of landmarks in the environment are given by $l = \{l^1, l^2, ..., l^n\}$. The landmarks are features in the environment and are not to be confused with the landmarks in heuristic planning where they are intended as a set of operators such that each plan must contain some element of this set. The goal is to reach a certain final state x_g with the localization uncertainty not greater than a given bound. Starting from an initial pose, the corresponding goal leading plan is to be synthesized, minimizing the makespan. We use the trace of the state covariance to quantify the uncertainty and the uncertainty condition is mathematically written as $\mathrm{Tr}(\Sigma_k) < \eta$. To incorporate belief evolution while planning, the DiNo planner is extended to support external calls evaluating the belief at each planning stage. The belief, the process and the measurement noise are assumed to be Gaussian.

Our TMP approach depends directly on the temporal planning horizon and the PDDL+ process discretization used. A longer horizon and shorter discretization increases planning complexity and directly affects the plan time. In Section 5.2, we evaluate the performance based on these factors and analyze how our approach cope with the changes in these elements.

4.2 Planner Workflow

An overview of our TMP planner framework is shown in Figure 1. We assume that the environment map, robot initial belief $\mathcal{N}(\mu_0, \Sigma_0)$, the goal pose to be reached x_g and the minimum pose certainty η required at goal are known. As discussed in the beginning of the Section, we utilize task planning to synthesize a plan, performing search in the motion space. Standard RRT based approaches sample waypoints that connects the start and end locations. However, to reduce pose uncertainty it is to be ensured that such connected paths have ample number of waypoints from which landmarks can be observed. To facilitate this perception-aware search we implement an RRT based potential field approach to sample such relevant waypoints in the environment. Waypoints near the potential field of the landmarks are pulled closer towards it and once a sufficient number (currently user defined) of such waypoints are generated, the further nodes are pushed away from the potential field. This sampled set of waypoints will be denoted as $wp = \{wp^1, ..., wp^m\}$.

The PDDL+ based event *belief update* triggers the semantic attachment call to the external library. The external library performs the belief updates (Eqs. 6, 7) attaching to the event effects the updated belief. Our semantic attachment effects guides the staged RPG (SRPG) construction in DiNo. The belief estimates returned by the semantic attachments guides the SRPG in identifying the *helpful actions* (e.g., the landmarks that can be visited in the next state), besides providing a heuristic function to select the next best landmark to visit. Based on the heuristics, a breadth first algorithm is performed to determine the forward state space leading to a state graph. The plan synthesized from the state graph such that the makespan is minimized by choosing least cost path that also satisfies all the constraints.

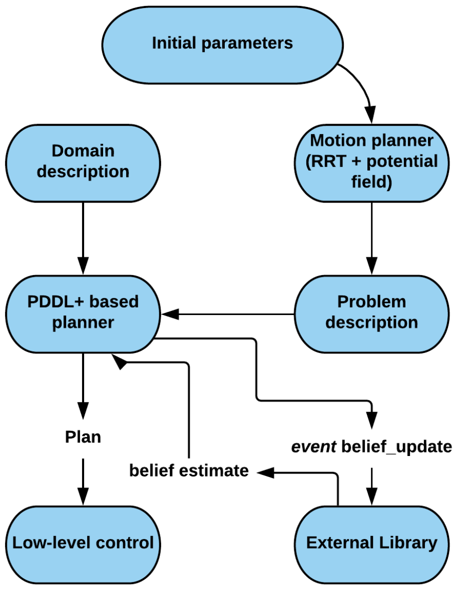


Figure 1: The TMP planner workflow.

4.3 External Calls in PDDL+

The PDDL+ description for our mobile robot scenario is shown in Figure 2. We extend the DiNo planner to incorporate semantic attachments, computing the propagated belief $b[\bar{X}_{k+1}]$ upon executing a control u_k at state x_k, as well as the posterior belief $b[X_{k+1}]$ upon obtaining a measurement. Since we are in the planning phase and yet to obtain observation, we simulate future observations z_{k+1} given the propagated belief $b[\bar{X}_{k+1}]$, the set of landmarks l and the measurement model given in Eq. (3. Given a pose $x \in b[\bar{X}_{k+1}]$, the nominal observation $\hat{z} = h(x, l^i)$ is corrupted with noise to obtain z_{k+1}.

Now we describe the formal algorithm for performing the belief updates. This procedure is summarized in Algorithm 1. The focal elements in the planning domain are: the action *goto_waypoint*, the event *belief_update* that triggers the external call to evaluate to perform the belief updates and the process *odometry* that simulates the robot motion between each planner discretization Δ. Starting from a given waypoint (line 2) the *goto_waypoint* action (line 3) initiates the robot motion towards a connected waypoint. The immediate effect of this action is to initialize the distance between the two waypoints (line 4) which starts the process *odometry* as seen in line 6. The process effect simulates the translational motion at each Δ and decreasing the distance between the waypoints by $\delta_{trans_k} = \Delta \times dFactor$ (line 7).

Event *belief_update* is immediately initiated (line 9) which triggers the semantic attachment call to perform belief update, returning the propagated belief $b[\bar{X}_{k+1}]$ to the event effect (line 9). If a landmark $l^i \in l$ is within the sensor range, the posterior belief $b[X_{k+1}]$ is evaluated returning its trace (line 11) to the event effect. To ensure the *process-event-process* ordering we employ a variable *counter* as shown in Figure 2.

Algorithm 1 Belief update implementation in PDDL+

Input: Set of waypoints wp, set of landmarks l, trace of initial pose covariance $\mathrm{Tr}(\Sigma_0)$, upper bound on the goal trace η, motion discretization factor $dFactor$, PDDL+ process discretization Δ

```
1:  while $\mathrm{Tr}(\Sigma_k) > \eta$ and $\neg$(robot_at $goalpose$) do
2:      (robot_at $wp\_from$)
3:      :action goto_waypoint
4:      $d(from,to)$ = distance($wp\_from$, $wp\_to$ )
5:      while $d(from,to) > -\delta_{trans_k}$ do
6:          :process odometry
7:          $d(from,to) \leftarrow d(from,to) - \delta_{trans\_k}$
8:          :event belief_update
9:          $\mathrm{Tr}(\bar{\Sigma}_{k+1}) \leftarrow \mathrm{Tr}(\Sigma_k)$            ▷ Eq. 6
10:         if landmark within sensor range then
11:             $\mathrm{Tr}(\Sigma_{k+1}) \leftarrow \mathrm{Tr}(\bar{\Sigma}_{k+1})$     ▷ Eq. 7
12:         end if
13:     end while
14:     :action reached
15:     (robot_at $wp\_from$) $\leftarrow$ (robot_at $wp\_to$)
16: end while
17: return Plan
```

5 Experimental Results

We evaluate our approach in a simple yet realistic experiment in the Gazebo simulator. Consider the corridor environment as seen in Figure 3 where the turtlebot robot starting from the initial waypoint (s in figure) needs to reach near the only plug point or the goal waypoint (g in figure) to recharge the batteries. The turtlebot is initially oriented towards g. Due to the short length of the charging cable, given the mean goal pose, there is a bound on the maximum pose uncertainty the robot can afford. The cubes marked 1-4 are the landmarks in environment. The *slam_gmapping* ROS package is used to build the environment map. The resulting map of the environment is shown in Figure 4a. The turtlebot with its laser scanner can be seen facing the goal waypoint marked in blue.

In the remainder of this section we discusses a number of test cases performed with the same starting and goal waypoints as shown in Figure 3. It is to be noted that for each given landmark we consider a potential field originating at its center. With the inclusion of the occupancy grid obtained during the mapping and the potential field, the RRT is constructed maintaining a closer proximity to the landmarks. The computational complexity depends exponentially on the number of waypoints m generated. The heuris-

```
(define (domain landmark)

(:requirements :typing :durative-actions :fluents :time
:strips
:disjunctive-preconditions :durative-actions
:negative-preconditions :timed-initial-literals)

(:types
                waypoint
                robot
                covariance
)

(:predicates
                (robot_at ?r - robot ?wp - waypoint)
                (visited ?wp - waypoint)
                (observe)
                (moving ?r - robot ?to - waypoint)
                (connected ?from ?to - waypoint)
                (lessthan ?c ?f - covariance)
)

(:functions
        (distance ?wp1 ?wp2 - waypoint) (cov) (counter)
        (update_covariance)     (predict_covariance)
        (relativeD) (dFactor) (finalTrace)
)
;; relativeD- a variable to store the distance
between state and wp as robot moves
;; dFactor \times #t - distance factor, to see
how many times kalman prediction to be done
;; cov is the initial covarianc trace,
finalTrace- the required trace upon reaching the goal state
;; action - Move between any two waypoints,
along the straight line between the two waypoints

(:action goto_waypoint
         :parameters (?r - robot ?from ?to - waypoint)
         :precondition (and (robot_at ?r ?from)
      (observe) (not(robot_at ?r ?to))
(not (visited ?to)) (connected ?from ?to))
         :effect (and
(not (robot_at ?r ?from))
      (assign (relativeD) (distance ?from ?to))
          (moving ?r ?to) (not (observe))
      (assign (counter) 0)
      (increase (update_covariance) 0)
      (increase (predict_covariance) 0))
)

(:action reached
                :parameters (?r - robot ?to - waypoint)
                :precondition (and (moving ?r ?to)
        (<= (relativeD) 0))
                :effect (and
(robot_at ?r ?to) (visited ?to)
        (not (moving ?r ?to) (observe))
)

(:event belief_update
                :parameters ()
                :precondition (and (> (counter) 0) )
                :effect (and
(assign (cov)
        (update_covariance)) (assign (counter) 0))
)

;; to calculate the number of update steps needed
(:process odometry
                :parameters ()
                :precondition (and
(> (relativeD) (-dFactor)) )
                :effect (and
(decrease (relativeD) (* #t (dFactor)))
                (increase (counter) (* #t 1)))
)
)
```

Figure 2: Domain description for the mobile robot scenario. The *process* odometry is used to simulate the robot translation and the *event* belief_update performs the belief propagation and posterior computation using semantic attachments.

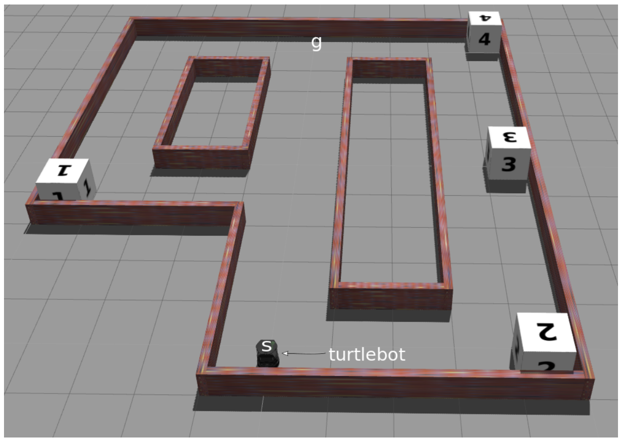


Figure 3: Corridor environment in Gazebo.

tic based search of the DiNo planner, reduces this state space explosion significantly. Furthermore, due to our potential field based RRT sampling, we are able to prune unwanted state expansions by generate parsimoniously connected waypoints which are sufficient for synthesizing satisficing plans.

5.1 ROSPlan Simulation

The initial variance in x, y, θ are 0.6m^2, 0.6m^2, 0.02rad respectively giving $\mathrm{Tr}(\Sigma_0) = 1.22$. PDDL+ process discretization of $\Delta = 1$ and a temporal horizon of $T = 20$ is used but with different values of η. Motion discretization factor $dFactor = 1$, giving $\delta_{trans_k} = 1$ (see Section 4.3). Performing the belief updates at each δ_{trans_k} helps in pruning the nearby waypoints and thereby reducing the state space explosion. However in the *difficult* regions where one cannot afford to prune close-by waypoints, the updates are performed upon reaching each of these waypoints. Unless otherwise mentioned, the number of waypoints for each case $m = 40$. The different test cases are detailed below.

Case 1: For this case $\eta = 0.3$. The plan generated is shown via the red path in Figure 4b, starting form initial state to the goal state. This is plan is quite expected due to the landmark rich nature of the path and the tight bound on the final trace. *Case 2*: For this case $\eta = 0.6$. The plan generated is the one following the red path shown in Figure 4c. *Case 3* has the same $\eta = 0.6$, however the path followed is along the one with 3 landmarks (Figure 4d). This is due to the randomness of the potential field based RRT motion planner and the bound on the trace being less tight. Since the objective function requires the planning time to be minimized, the planner returns the path which minimizes the makespan which in turn is determined by the waypoint locations in the environment. As seen in Table 1, the states expanded and the planning time is much larger than in *Case 2* (around 4.8 times). This is due to the large number of connected waypoints. *Case 4* has $\eta = 0.6$ and the path followed (see Figure 4e) is similar to the one in *Case 3*. However, as evident from Figures 4e, in this case the number of connected waypoint is much less, resulting in a much lesser planning time (see Table 1). *case 5* (see Figure 4f) is similar to the *case 2* but with differences in the number of connected waypoints. This is reflected in the states expanded and the planning times as seen in Table 1.

Cases 6,7 and *8* have $\eta = 0.9$. However as discussed previously the sampling of waypoints affects the plan as can be seen in Figures 4g- 4i. In Figure 4g the plan generated is such that the trajectory upon reaching a waypoint near the goal, is extended to a waypoint near landmark 4, from which it can be observed. *Case 8* has the highest number of connected waypoints and this is reflected in the corresponding row in Table 1.

Case 9 is similar to *Case 1* but with higher number of connected waypoints. In *Case 10*, $m = 20$ and $\eta = 0.6$. The reduction in waypoint significantly reduces the state space explosion and the planning time, as can be seen in the last row of Table 1. However it can be seen in Figure 4k that the upper path has no waypoints sampled. Though it is true that for our all the cases discussed so far this doesn't cause any alarm, in general a lesser number of waypoints might resulting in no plans being produced, even though there exists one. For example, changing the starting or the gaol location in our experiments can lead to a path via landmark 1. Finding such a minimum number of waypoints is another planning problem by itself.

Test cases	States expanded	Plan time (s)
Case 1	6184	116.62
Case 2	6934	156.62
Case 3	33769	744.36
Case 4	2710	60.64
Case 5	4847	99.74
Case 6	7646	190.44
Case 7	7289	143.04
Case 8	56544	1119.00
Case 9	28595	684.60
Case 10	388	4.60

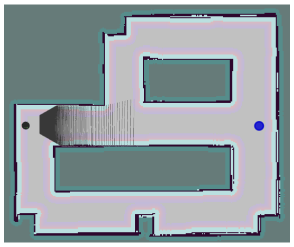
Table 1: Different test case with number of states expanded in each case and the corresponding planning time in seconds.

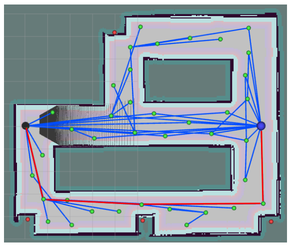
It is to be noted that in *Cases 2, 5, 7 and 10*, the robot might collide with the walls due to the initial covariance increase. This shows that a motion plan alone fails and logical reasoning of action effects is required in conjunction with motion planning. For example, rather than just the heuristic search, an A* algorithm incorporating the $\mathrm{Tr}(\Sigma_g)$ in $g(\cdot)$ would prevent the generation of such flawed plans.

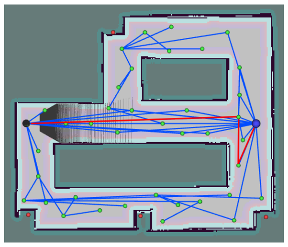
5.2 Performance Analysis

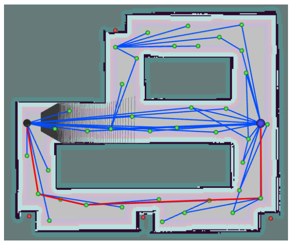
To analyze the performance based on the temporal planning horizon and the PDDL+ process discretization we perform different simulations using *Case1* in Section 5.1 as our base case. We use different discretization Δ and time horizon T and analyze the states explored, computation/plan time t and the goal covariance trace $\mathrm{Tr}(\Sigma_g)$.

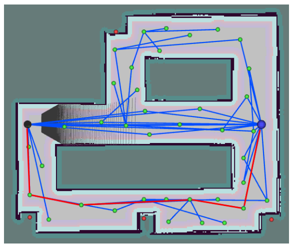
From table 2 we can observe that the planning time is inversely proportional to the discretization Δ. This is because we perform belief updates based on $\delta_{trans_k} = \Delta \times dFactor$.

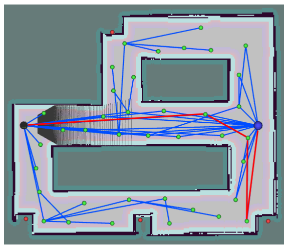

(a) Initial configuration

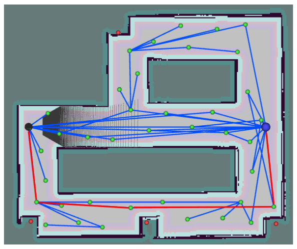

(b) Case 1

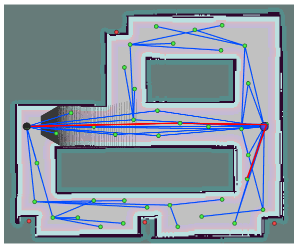

(c) Case 2

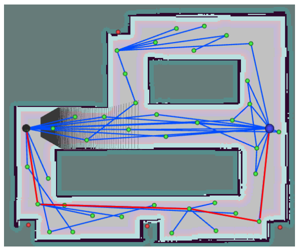

(d) Case 3

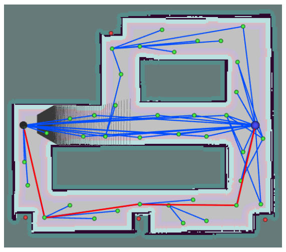

(e) Case 4


(f) Case 5


(g) Case 6


(h) Case 7


(i) Case 8


(j) Case 9

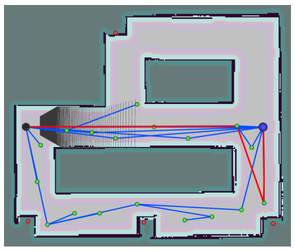

(k) Case 10

Figure 4: Initial configuration and the trajectories for the different test cases.

Δ	T	States expanded	t (s)	$\mathrm{Tr}(\Sigma_g)$
0.5	20	47500	1231.60	0.076
0.5	25	112601	N/A	N/A
0.5	30	112601	N/A	N/A
0.5	35	112601	N/A	N/A
1.0	20	6184	121.88	0.153
1.0	25	12788	249.14	0.156
1.0	30	23533	566.44	0.143
1.0	35	41682	1092.96	0.181
2.0	20	750	17.56	0.267
2.0	25	1290	39.66	0.273
2.0	30	995	37.98	0.290
2.0	35	967	47.78	0.255

Table 2: Performance parameters for different temporal planning horizons and PDDL+ process discretization. N/A-planner ran out of memory.

Since $dFactor = 1$, the number of updates is directly proportional to Δ, thereby increasing the states expanded and hence the plan time. It is seen that the temporal horizon and total states searched are correlated, directly affecting the total plan time. It can be observed from the Table that for a longer temporal horizon T and a larger Δ, the goal condition is satisfied by expanding fewer states. Larger the value of T, the more deeper SRPG is built resulting in better heuristic. However, for 2 given waypoints, lower value of Δ implies additional belief updates, leading to more states being expanded. Conversely, lower discretization and a larger time horizon will not result in successful plans. Longer T can lead to more states being expanded and a large Δ can lead to unvalidated plans. Hence an efficient plan requires a trade-off between the two.

6 Conclusion

We have discussed an ongoing research for mobile robot localization in the area of TMP, equipping a hybrid task planner with the capability of reasoning in the belief space of the robot. Expressive power of PDDL+ combined with heuristic base semantic attachments simulate the belief evolutions given an action sequence and the corresponding expected future observations. The underlying methodology of the hybrid planner has been discussed, validating the approach using a realistic synthetic simulation in Gazebo. We also provide the first step towards perception aware TMP, by sampling information rich waypoints and reasoning about landmark observations from these waypoints in the planning phase. Our sampling strategy combined with motion discretization help reduce the state space explosion while planning.The TMP planner is also unified with the ROSPlan framework.

While the scalability to larger domains still remains a challenge, exploiting the planning-as-model-checking nature of the of the DiNo planner along withing efficient caching of plans might help in tackling this issue to some extent. Effective sampling strategies can help in pruning the unwanted state expansion. The extant of such pruning needs to be studied in detail. At present, DiNo supports only the minimize *total-time* metric and hence the plan synthesized is the one with minimum makespan. In our experiments the goal condition required only a bound on the trace of the final covariance. Yet, most often we require different costs to be minimized throughout the plan and hence we are working towards incorporating the same. We agree that in this paper we discuss in detail the planning algorithm and not the low-level motion control for execution and leave it for future work. However, we have successfully tested the execution for all our experiments in ROSPlan.

References

Bernardini, S.; Fox, M.; Long, D.; and Piacentini, C. 2017. Boosting Search Guidance in Problems with Semantic Attachments. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 29–37.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research* 28(1):104–126.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *International Conference on Automated Planning and Scheduling*, 333–341.

Dantam, N. T.; Kingston, Z. K.; Chaudhuri, S.; and Kavraki, L. E. 2018. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research* 0(0):0278364918761570.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009. Integrating symbolic and geometric planning for mobile manipulation. In *Safety, Security & Rescue Robotics (SSRR), IEEE International Workshop on*, 1–6. IEEE.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2012. Semantic Attachments for Domain-Independent Planning Systems. In *Towards Service Robots for Everyday Environments*. Springer Berlin Heidelberg. 99–115.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3-4):189–208.

Fox, M., and Long, D. 2006. Modelling Mixed Discrete-Continuous Domains for Planning. *Journal of Artificial Intelligence Research* 27(1):235–297.

Fox, D.; Burgard, W.; and Thrun, S. 1997. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics & Automation Magazine* 4(1):23–33.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research* 20:291–341.

Kaelbling, L. P., and Lozano-Pérez, T. 2012. Integrated robot task and motion planning in the now. Technical Report 2012-018, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *The International Journal of Robotics Research* 32(9-10):1194–1227.

Kavraki, L. E.; Svestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation* 12(4):566–580.

Kavraki, L. E.; Kolountzakis, M. N.; and Latombe, J.-C. 1998. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation* 14(1):166–171.

Kuffner, J. J., and LaValle, S. M. 2000. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, 995–1001. IEEE.

Lozano-Pérez, T., and Kaelbling, L. P. 2014. A constraint-based method for solving sequential manipulation planning problems. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 3684–3691. IEEE.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL- The Planning Domain Definition Language. In *AIPS-98 Planning Competition Committee*.

Nilsson, N. J. 1984. Shakey the robot. Technical Report 323, Airtificial Intellignece Center, SRI International, Menlo Park, California.

Phiquepal, C., and Toussaint, M. 2017. Combined task and motion planning under partial observability: An optimization-based approach. *RSS Workshop on Integrated Task and Motion Planning*.

Piotrowski, W. M.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2016. Heuristic Planning for PDDL+ Domains. In *AAAI Workshop: Planning for Hybrid Systems*.

Srivastava, S.; Fang, E.; Riano, L.; Chitnis, R.; Russell, S.; and Abbeel, P. 2014. Combined task and motion planning through an extensible planner-independent interface layer. In *Robotics and Automation (ICRA), IEEE International Conference on*, 639–646. IEEE.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic Robotics*. MIT press.

Toussaint, M. 2015. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Weyhrauch, R. W. 1980. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence* 13.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Risk-Aware Planning by Extracting Uncertainty from Deep Learning-Based Perception

Maymoonah Toubeh and Pratap Tokekar
Virginia Polytechnic Institute and State University
Bradley Department of Electrical and Computer Engineering
Blacksburg, Virginia 24061

Abstract

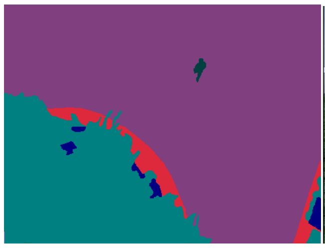
The integration of deep learning models and classical techniques in robotics is constantly creating solutions to problems once thought out of reach. The issues arising in most models that work involve the gap between experimentation and reality, with a need for a quantification of risk in real-world situations. In order to translate advances in robot planning techniques that use deep learning to safety-critical applications, strategies must be developed and applied to assess the risk involved with different models. This work proposes the use of Bayesian approximations of uncertainty from deep learning in a robot planner, showing that this produces more cautious actions in safety-critical scenarios. An example setup involving a deep learning semantic image segmentation, followed by a path planner based on the resulting cost map, is used to provide an empirical analysis of the proposed method.

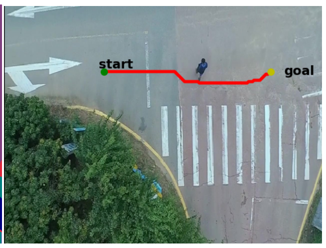
Introduction

Recent advances in deep learning (DL) training algorithms, paired with significant improvements in hardware, have shown potential in many fields, including robotics. From enabling robot systems to navigate using high-dimensional image inputs, to allowing tractable trial-and-error robot learning both in simulation and reality; deep learning is everywhere. However, as promising as applications of DL to robot planning seem, the potential of the positive impact they may have on real-world scenarios is inevitably proportionate to their interpretability and applicability to imperfect environments.

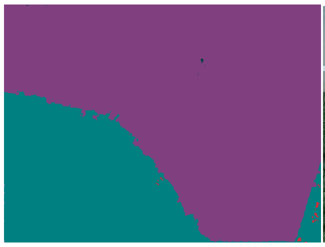
For an example setup, this work utilizes a DL image segmentation model to generate a cost map used in an A* path planner. Figure 1 shows qualitative results given by the DL model and the subsequent planner. In this image taken from the Aeroscapes dataset (Nigam, Huang, and Ramanan 2018), the pedestrian near the top center is not sufficiently segmented by the DL model prediction shown in Figure 1a. Incorporating uncertainty associated with this prediction before passing it to the planner produces a more reasonable and risk-aware resulting path, as seen in Figure 1f. Higher levels of uncertainty are visualized by darker spots in Figure 1e.

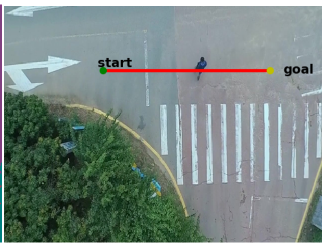
DL is known for its data-driven rather than algorithmic learned representations (LeCun, Bengio, and Hinton 2015). A DL hierarchical structure can learn directly from data with


(a) Handcrafted ground truth segmentation

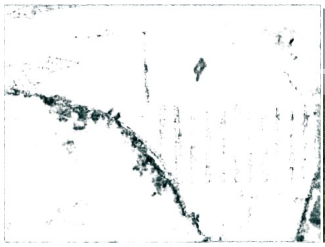


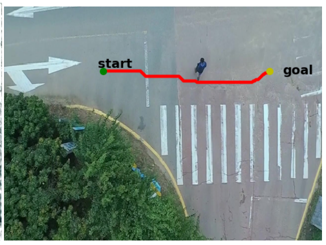
(b) Planning based on ground truth segmentation


(c) DL model segmentation



(d) Planning based on DL model segmentation alone


(e) Uncertainty of DL model segmentation



(f) Planning based on DL model segmentation with uncertainty

Figure 1: Qualitative results showing the path planned from start to goal given (a) handcrafted ground truth image segments, (b) deep learning model segmentation alone, (c) deep learning model segmentation with uncertainty.

little to no handcrafted features or learning variables (Goodfellow, Bengio, and Courville 2016). However, this often comes at the expense of the interpretability of the learning outcomes. Deep neural networks can even misrepresent data outside the training distribution, giving predictions that are incorrect without providing a clear measure of certainty associated with the result (Gal 2017). The outputs of a deep neural network are generally point estimates of the parameters and predictions present, so they do not provide a meaningful measure of correlation to the overall data distribution the network was trained on (Gal 2017). For this reason, deep learning models are considered deterministic functions, often called "black-boxes", unlike probabilistic models which inherently depict uncertainty information.

As a step towards risk-aware robotic systems that utilize the powers of DL, this work combines methods of approximating uncertainty in DL with robot planners that are partially reliant on an otherwise black-box approach. We utilize modern methods of uncertainty extraction from deep learning models, specifically those that do not interfere with the overall structure or training process (Gal and Ghahramani 2016). The extraction of uncertainty information, as opposed to the reliance on point estimates, is crucial in safety-critical applications, such as autonomous navigation in an urban setting. If a robot planner encounters out-of-distribution data at test time, it is preferable that the system provides an uncertainty metric to allow for more meaningful interpretation of a forced point estimate. With the acquired Bayesian uncertainty estimates, the system produces an explicable metric and can therefore be altered to accommodate for risks in the environment.

In the robotics community, an emphasis has been placed on methods that work in a controlled experimental setup, but more recently risk-aware methods aim to ensure that these methods are also safe in the real-world. As a relatively new application to robotics, techniques have been adapted from the fields of statistics and machine learning. Common statistical methods of accommodating risk include altering the optimization criterion so that it becomes risk-sensitive (Garca and Fernández 2015). Although modern DL models are usually considered black-boxes due to their mathematical nature, recent work has initiated theoretically grounded understandings of them. Such works investigate the integration of deep learning techniques with information theoretical and statistical approaches for the purpose of calculating model uncertainties (Gal 2017). It is these practical methods of quantifying risk associated with DL models that are utilized in the proposed planning systems of this paper.

Related Work

Several prior works exist which have extracted uncertainty information from a deep learning model, mainly Bayesian neural networks, ensemble methods, and methods that utilize stochastic regularization techniques (Gal 2017; Osband et al. 2016). These works have also produced meaningful contributions to real-world applications, such as gas turbine control, camera relocalization, and robotic collision avoidance (Depeweg et al. 2016; Kahn et al. 2017; Kendall and Cipolla 2016). Our work seeks to extend the most suitable of these approaches to robot planning.

Bayesian Neural Networks

Some of the earliest attempts to bind the reasoning of probabilistic models, such as Gaussian processes, with deep learning (DL) is seen in Bayesian neural networks (BNN) (Gal 2017; Kononenko 1989). Unlike the DL models used in modern practice which do not depict uncertainty, a BNN produces an output that is a probability distribution over its predictions. Probability distributions are placed over the weights of a BNN, making it an approximation of a Gaussian process as the number of weights tends to infinity. Uncertainty can be extracted as a statistical measure, such as variance or entropy, over this output distribution in order to capture how confident the model is with its prediction. However, BNN require a larger number of parameters to be trained, with less practical training methods available for them. A recent example of a Bayesian neural network applied to a stochastic dynamic system utilizes a smaller model and trains by minimizing alpha-divergences (Depeweg et al. 2016). The system dynamics are learned using the BNN and are fed into a model-based reinforcement learner for control, with application to a gas turbine.

Ensemble Methods

Since the practicality of Bayesian neural networks is questionable, approximations of these structures have arisen, including ensemble methods. One such recent method is referred to as the bootstrapped neural network, where several deep learning (DL) models are trained on subsets of the larger dataset sampled with replacement. The underlying concept behind this method is that the different models will agree in high density areas and disagree in low density areas of the complete dataset. The outputs of the separate models combine to form a probability density function from which uncertainty of a particular prediction can be measured. This method is theoretically sound; however, it is not ideal for applications that are faced with time and resource constraints, such as robotics.

A recent work proposes using bootstrapped deep Q-learning networks quantifying uncertainty to direct the learning process towards more efficient exploration, which is a key issue in reinforcement learning (Osband et al. 2016). The technique involves sampling the past experiences in Q-learning, rather than taking the whole sequence, in order to form an estimate of the Q-value at a given state. Sampling is also meant to scale better to large state-spaces. A similar approach is used to improve a form of Q-learning, Deep Deterministic Policy Gradient (DDPG) (Kalweit and Boedecker 2017). The work demonstrates that using bootstrapped uncertainty to direct data collection produces faster learning that is also less expensive, tedious, or likely to lead to physical damage for a real robot.

Stochastic Regularization Methods

Most recently, the use of stochastic regularization methods common in deep learning (DL) has been shown to also

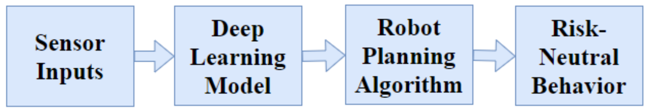
Class	Sky	Building	Pole	Road	Pavement	Tree	Sign Symbol	Fence	Car	Pedestrian	Bicyclist	Unlabeled
Cost	15	11	10	1	2	7	9	8	12	14	13	16
Color	⬜	🟦	🟦	🟪	🟥	🟩	🟪	🟫	🟥	🟩	🟧	⬛

Table 1: The fixed costs assigned to each segmentation to be used in A* search. These costs are hand-designed to generate qualitative examples that demonstrate the utility of the proposed approach. In the real world, classes that are not navigable (e.g., fence, car, bicylist) will be assigned infinite cost.

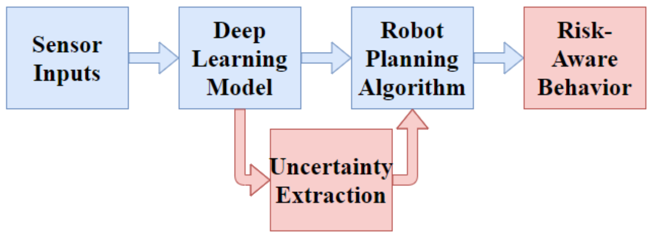
approximate Bayesian neural network models without any changes to the ready structures being used or their training process (Gal 2017). Regularization methods are used as a means to avoid overfitting of a DL model to its training set, so that it generalizes to data that is similar enough but not exactly the same. This ensures the learning model has not simply memorized the training data, but has actually learned something meaningful about the data that will translate to a slightly different setting. At a high level, stochastic regularization techniques work by introducing randomness in the training process to increase the robustness of the model to noise. Dropout is one such popular method that is inspired by the probabilistic interpretations of deep learning models that consider activation nonlinearity a cumulative distribution function (Bishop 2006). In its traditional use, dropout is activated during training, in which case the weights of a deep learning model are randomly multiplied by zero or one in a certain predefined proportion. In this approach, at test time, weight averaging by the percentage of dropout applied during training is performed on the trained model, which then leads to point estimate results.

In order to form a distribution over the outputs of a model trained using a stochastic regularization technique such as dropout, the regularization is activated at test time, producing stochastic estimates with multiple passes of the same input through the model (Gal and Ghahramani 2016). The multiple stochastic passes are then averaged to form a mean estimate, as opposed to a point estimate, and uncertainty can be extracted given the statistical or information theoretical metrics over the distribution.

One recent work applies Bayesian approximation using dropout to a pre-existing model-based reinforcement learning algorithm, called PILCO, in order to better quantify uncertainty over longer periods of learning (Gal, McAllister, and Rasmussen 2016). The authors replace the original Gaussian processes with a deep neural network. In another work, the Bayesian approximation of uncertainty with dropout is used to assist in camera relocalization for landmark detection in a SLAM problem (Kendall and Cipolla 2016).The uncertainty estimate is used to approximate the localization error with no additional hand-crafted parameterizations. In a similar work, the same Bayesian approximation approach to uncertainty is applied to semantic segmentation for improved learning and test time estimation (Kendall, Badrinarayanan, and Cipoll 2015). In yet another work, the authors combine bootstrapped neural networks with stochastic regularization methods to avoid catastrophic or harsh collisions during robot training for collision avoidance (Kahn et al. 2017). They show that their method effectively minimizes dangerous collisions during training, while also showing comparable performance to baselines without explicit account for uncertainty. A more recent approach proposes the use of a Mixture Density Model (MDM) as a replacement for the stochastic passes, where a single pass saves time in comparison to multiple in a robotics setting (Choi et al. 2017). However, this comes at the expense of training a separate MDM network for the task of uncertainty extraction.



(a) Risk-Neutral Perception



(b) Risk-Aware Perception

Figure 2: The overall flow of information in a robot planner that relies in part on a DL model in (a) the risk-neutral and (b) the risk-aware case.

Problem Formulation

The problem setup is inspired by a previous work (Christie et al. 2017) that uses the overhead imagery provided by an unmanned aerial vehicle (UAV) as input to an image segmentation algorithm, which is then used to assist the navigation of an unmanned ground vehicle (UGV). The UAV acts as a “scout” by flying ahead of the UGV. The overhead orthorectified imagery is then classified (into categories such as “road”, “pavement”, “car”, etc.). Each category is assigned a cost (given in Table 1) which is used to determine a path for the ground robot to follow.

Figure 2 shows a high-level schematic of the proposed risk-aware approach by contrasting it with the traditional, risk-neural approach. Unlike the previous work, here, in addition to performing the semantic segmentation of the image, the uncertainty in the segmentation is also extracted. The measure of uncertainty is then used to manipulate the navigation away from low confidence regions. The navigation portion of the robot planner in this case is not a DL model, but a classical method, A* search. A cost function is mapped onto each semantic class (see Table 1). Therefore, the uncertainty in the segmentation corresponds to uncertainties in the cost, for which A* search is sufficient. If the uncertainties in the transition function are to be considered instead, a Markov Decision Process (MDP) would be more useful. Considering uncertainty is contrast to trusting the outputs of the DL portion of the system invariably, which could lead to catastrophic outcomes if a point estimate outlier is produced in the case the input is considered out-of-distribution to the training data. In the proposed approach, an uncertainty metric can be used to calibrate the robot plan based on the level of confidence in the DL model predictions.

The results of the segmentation given by any DL model cannot guarantee complete accuracy in all settings. Variations in lighting, angle, or objects present in an image can contribute to inaccurate predictions. There will always be a prediction when a DL model is involved, as the model will force an estimate even when it does not make sense to. A good measure to account for this risk associated with DL outputs being used in the robot planner is to evaluate the certainty associated with the DL result. One practical method is using dropout, which is already being used as a regularizer during training.

In the risk-neutral case, the pixel classification is taken as is from the DL model and assigned a cost accordingly. For the proposed risk-aware method, the cost is evaluated by adding the uncertainty value, multiplied by some factor, to the risk-neutral cost assignment. Specifically, the cost of pixel p is given by,

$$C(p) = L(p) + \lambda \hat{V}(p), \tag{1}$$

where $L(p)$ is the cost associated with the semantic class that is predicted for p (given in Table 1) and $\hat{V}(p)$ is the uncertainty value extracted by dropout. In practice, $\hat{V}(p)$ does not need to be variance; it can be another statistical measure of uncertainty taken over the distribution provided by the stochastic passes. λ is a weighting parameter. The risk-neutral case corresponds to $\lambda = 0$ and the risk-aware case corresponds to $\lambda > 0$.

Algorithm 1 shows a breakdown of uncertainty extraction given an input image. First, the stochastic outputs are generated for a number of stochastic passes, giving a softmax value for each pixel each time. For each pass, Bernoulli dropout is activated on the trained network, effectively multiplying random neuron weights by zero or one in a set proportion. The softmax outputs O are averaged over all stochastic passes. The maximum value of this average softmax is taken as the output class prediction Ind. Variance is computed over the stochastic passes for each output class, then the average of V over all classes produces a single value for each pixel. The value $\hat{V}$ is considered the uncertainty in that pixel's prediction.

Algorithm 1 Uncertainty Extraction

Input: image I
Output: class of average prediction for each pixel $Ind(p)$, average variance for each pixel $\hat{V}(p)$

```
 1: $p \leftarrow$ pixels in image $I$
 2: for $t$ = 1 to number of stochastic passes do
 3:     $O(p, c, t) \leftarrow$ softmax output of stochastic pass $t$
 4: end for
 5: $\hat{O}(p, c) \leftarrow$ average $O(p, c, t)$ over stochastic passes
 6: $Ind(p) \leftarrow$ argmax of softmax in $\hat{O}(p, c)$
 7: for $c$ = 1 to number of classes do
 8:     $V(p, c) \leftarrow$ variance in $O(p, c, t)$ over stochastic passes for class $c$
 9: end for
10: $\hat{V}(p) \leftarrow$ average $V(p, c)$ over classes
```

The uncertainty value is computed as the average variance across all segment classes for a particular pixel. The higher the average variance, the less confident the DL model is with its prediction. Therefore, it is intuitive to incorporate this information along with the original prediction when planning a path for navigation, especially in a safety-critical environment such as a road.

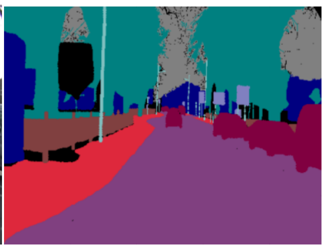
Experimental Setup

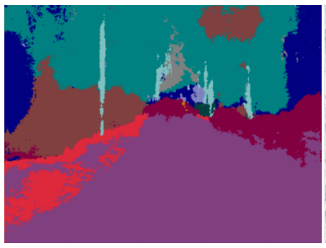
To demonstrate the utility of an uncertainty metric associated with a DL model being used as part of a robot planning system, experiments involving this setup are portrayed to compare the risk-neutral and risk-averse cases qualitatively.

We use the Bayesian SegNet to perform semantic segmentation of every pixel in the input image (Kendall, Badrinarayanan, and Cipoll 2015). The Bayesian SegNet is first trained for the segmentation task using the predefined model architecture, along with the pretrained weights of the VGG16 image classification network (Kendall, Badrinarayanan, and Cipoll 2015). Since the model is already well adapted for image classification, less further training is needed for per-pixel segmentation, in comparison to starting with random model weighting. The CamVid dataset (Brostow, Fauqueur, and Cipolla 2009) of 367 training and 233 testing images of road scenes is used. The model converges with a test accuracy of 94.56 %. A batch size of 2 is used to fit an 8GB Nvidia GeForce GTX 1080. Some results of this network are shown in Figure 3 for the CamVid test set, as well as for an entirely different road dataset called KITTI (Geiger et al. 2013) in Figure 4. It is worth noting that the KITTI dataset is an example for which the DL model is not explicitly trained, but its images are similar enough to the training set to expect reasonable segmentation performance. Errors are common, and expected, in a varying or real-world setting, but the uncertainty metric should provide a means of detecting such errors.

After the DL model is trained to produce reasonable results on images similar to the training set, the next step involves using the pixel segmentations as input to the A*

(a) Input road image


(b) Handcrafted ground truth segmentation


(c) DL model segmentation

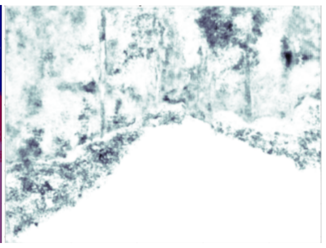
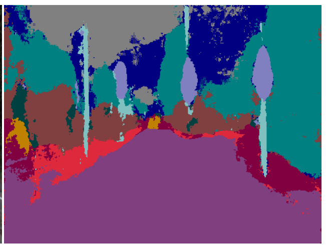

(d) DL model uncertainty

Figure 3: An example semantic segmentation produced by the Bayesian SegNet trained model on (a) an input image from the CamVid test set with (b) its handcrafted ground truth segmentation, alongside (c) the resulting output segmentation and (d) the model uncertainty associated with the output, with darker regions signifying higher uncertainty.

(a) Input road image


(b) DL model segmentation

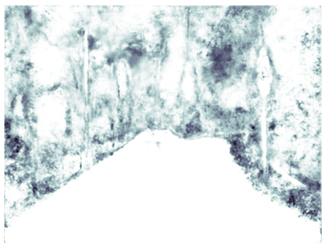

(c) DL model uncertainty

Figure 4: An example semantic segmentation produced by the Bayesian SegNet trained model on (a) an input image from the KITTI dataset with (b) the resulting output segmentation and (c) the model uncertainty associated with the output, with darker regions signifying higher uncertainty.

Measure	Ground Truth	Risk-Neutral	Risk-Aware
Expected Cost	251	234	253
Actual Cost	251	413	281
Surprise Factor	0	179	28

Table 2: The surprise factor calculated as the difference between the expected cost and actual cost for the example in Figure 1, given ground truth, risk-neutral, and risk-aware segmentation.

search planner. In order to use semantic segmentation in robot planning, a cost function must be defined over the segmentation result to be used in the planning. The most common method of translating segment identity into a real number cost involves some quantification of the segmentation class traversability combined with the robot's speed. For example, in the previous work inspiring our problem setup, segment traversability is proportionate to the power consumption of the UGV based on prior experience (Christie et al. 2017). However, for simplicity, here, the costs of the segment classes are set to a fixed value, as shown in Table 1. Once a cost is assigned, a start position and a goal position are defined in the input image, and the planning algorithm produces a path based on the cost assigned to the predicted pixel identification.

Results and Discussion

In order to perform a qualitative analysis of the risk-neutral and risk-aware methods, the *surprise factor* is calculated to compare the expected path cost with the actual path cost by subtracting the two. The expected path cost is found by summing up the cost associated with every pixel along the path. We use the predicted class of every pixel to determine the expected cost and the ground truth class of every pixel to determine the actual cost. If a path passes through pixels that the DL model classifies with low uncertainty, then we expect the predicted classes to be largely the same as the actual cost, thereby given a low surprise factor. On the other hand, if the predicted classes are wrong, then the surprise will be high.

Table 2 shows this factor in three planning scenarios of Figure 1: using ground truth segmentation, using DL model segmentation alone, and using DL model segmentation while taking into account its prediction uncertainty. When using DL segmentation alone, not accounting for model confidence increases the chance for a higher disparity between the expectation and reality, as seen in our example. On the other hand, although the risk-aware approach results in longer paths, the expectation better matches the reality and leads to lower surprise.

Table 3 shows a similar trend for the average surprise factor calculated over 100 randomly chosen starting and goal positions. In this case, the risk-neutral strategy tends to underestimate the actual path cost resulting in a larger surprise factor. On the other hand, the risk-aware strategy conservatively chooses longer paths but gives a significantly lower surprise factor.

Measure	Ground Truth	Risk-Neutral	Risk-Aware
Expected Cost	1221.60	1160.10	1427.40
Actual Cost	1221.60	1480.10	1405.35
Surprise Factor	0.00	320.00	22.05

Table 3: The average surprise factor calculated as the difference between the expected cost and actual cost for 100 randomly chosen starting and goal positions, given ground truth, risk-neutral, and risk-aware segmentation.

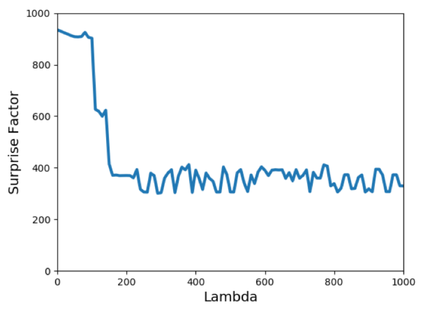


Figure 5: Effect of varying λ on the surprise factor.

Figure 5 shows the effect of varying the λ parameter on the surprise factor. When λ is small, the surprise factor is large. This is consistent with previous findings, since $\lambda = 0$ corresponds to the risk-neutral case. As λ increases, the surprise factor decreases finally converging to a fixed value. This is because, once λ is sufficiently large, increasing λ further does not change the path produced as output significantly (except for a few pixels). In fact, for very large λ, the path found will be the minimum uncertainty path since the second term in Equation 1 dominates the first term. Therefore, the surprise factor remains largely the same.

Conclusions and Future Work

This work proposes a risk-aware approach to robot planning that already involves deep learning. Risk is quantified by the model prediction uncertainty in the planning process. When deep learning is used for perception as a portion of the planning loop, an understanding of confidence in DL estimates is useful. Uncertainty is extracted directly from the DL models utilizing dropout as a practical method, especially in resource constrained settings such as robotics. Promising results show that including uncertainty in a planner provides better predictability of actions, and even the avoidance of catastrophic actions in a safety-critical setting. Further experiments and empirical analysis are to be conducted using statistically significant data to explore the potential of this method for real-world systems in the long run. Some scenarios include those presented in this paper, such as the UAV assisted navigation of a UGV using DL image segmentation, on hardware and photorealistic simulators. The main anticipated extensions of this work involve the use of a larger dataset analysis, along with investigating different uncertainty metrics extracted from the same experiment setup presented in this paper.

Acknowledgements

The material is based upon work supported by the National Science Foundation under Grant number 1566247.

References

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. New York: Springer.

Brostow, G. J.; Fauqueur, J.; and Cipolla, R. 2009. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters* 30(2):88–97.

Choi, S.; Lee, K.; Lim, S.; and Oh, S. 2017. Uncertainty-aware learning from demonstration using mixture density networks with sampling-free variance modeling. *arXiv preprint arXiv:1709.02249*.

Christie, G.; Shoemaker, A.; Kochersberger, K.; Tokekar, P.; McLean, L.; and Leonessa, A. 2017. Radiation search operations using scene understanding with autonomous uav and ugv. *Journal of Field Robotics* 34(8):1450–1468.

Depeweg, S.; Hernández-Lobato, J. M.; Doshi-Velez, F.; and Udluft, S. 2016. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*.

Gal, Y., and Ghahramani, Z. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, 1050–1059.

Gal, Y.; McAllister, R.; and Rasmussen, C. E. 2016. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*.

Gal, Y. 2017. *Uncertainty in Deep Learning*. Ph.D. Dissertation, University of Cambridge.

Garca, J., and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. In *Journal of Machine Learning Research*, volume 16, 1437–1480.

Geiger, A.; Lenz, P.; Stiller, C.; and Urtasun, R. 2013. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research*.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*, volume 1. Cambridge, MA: MIT Press.

Kahn, G.; Villaflor, A.; Pong, V.; Abbeel, P.; and Levine, S. 2017. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*.

Kalweit, G., and Boedecker, J. 2017. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, 195–206.

Kendall, A., and Cipolla, R. 2016. Modelling uncertainty in deep learning for camera relocalization. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, 4762–4769. IEEE.

Kendall, A.; Badrinarayanan, V.; and Cipoll, R. 2015. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*.

Kononenko, I. 1989. Bayesian neural networks. *Biological Cybernetics* 61(5):361–370.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521(7553):436.

Nigam, I.; Huang, C.; and Ramanan, D. 2018. Ensemble knowledge transfer for semantic segmentation. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 1499–1508. IEEE.

Osband, I.; Blundell, C.; Pritzel, A.; and Van Roy, B. 2016. Deep exploration via bootstrapped dqn. *Advances in Neural Information Processing Systems* 4026–4034.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Policy Networks for Reasoning in Long-Term Autonomy

Kyle Hollins Wray and **Shlomo Zilberstein**
College of Information and Computer Sciences, University of Massachusetts, Amherst, MA, USA

Abstract

Policy networks are graphical models that integrate decision-making models. They allow for multiple Markov decision processes (MDPs) that describe distinct focused aspects of a domain to work in harmony to solve a large-scale problem. This paper presents the formalization of policy networks and their use in modeling reasoning tasks necessary for scalable long-term autonomy. We prove that policy networks generalize a wide array of previous models, such as options and constrained MDPs, which can be equivalently viewed as the integration of multiple models. To illustrate the approach, we apply policy networks to the challenging real world domain of robotic home health care. We demonstrate the benefits of policy networks on a real robot and show how they facilitate scalable integration of multiple decision-making models.

Introduction

Over the past decade, sequential decision-making models have been increasingly deployed in large-scale domains with high societal impact, ranging from aircraft collision avoidance (Kochenderfer 2015) to autonomous vehicles (Wray, Witwicki, and Zilberstein 2017). While these systems have enjoyed rapid growth, they relied on a fragmented collection of specialized approaches that combine either multiple objectives (Altman 1999; Klein et al. 2012) or multiple models by hierarchical abstraction (Sutton, Precup, and Singh 1999; Pineau et al. 2003) or by integrating their actions online (Bai et al. 2015; Wray, Witwicki, and Zilberstein 2017).

Each one of these solutions introduces an important reasoning capability, but to support long-term autonomy in the real world, we increasingly need to integrate multiple capabilities within one system. As Marvin Minsky observed, "the power of intelligence stems from our vast diversity, not from any single, perfect principle" (Minsky 1986). It is unlikely that any single MDP model will suffice. For the sake of scalability and computational efficiency, we need new formal ways to facilitate the integration of multiple models within a single agent. To this end, we propose a novel framework called policy networks that unifies prior approaches, offers new insights, and provides a solid foundation on which to build the next generation of large-scale decision models.

We consider a home healthcare robot domain for household and eldercare scenarios (Pineau et al. 2003). The robot must perform a wide array of helpful tasks (e.g., medicine delivery and cleaning), plan safe paths around the house, and detect falls to call for help as needed (Broadbent et al. 2009) while operating over a long duration. This domain has many subproblems, each complex and nuanced, and they are all interrelated as part of the whole solution. Systems of this scale require an integration of many methods, as they quickly become too large to solve with a single monolithic MDP.

Prior work on integrations of multiple models arose from disparate ideas, each of which extends the MDP in a particular way. For hierarchies, a large problem is decomposed into essentially subtasks (Sutton, Precup, and Singh 1999; Tao et al. 2009). For multiple objectives, which we show is related, solutions typically scalarize the objectives into one or maximize a primary objective subject to constraints (Roijers, Whiteson, and Oliehoek 2014; Klein et al. 2012). For online solutions with multiple models, each update their state spaces simultaneously and recommend actions for each entity in the domain (Kochenderfer 2015; Bai et al. 2015).

While these approaches have been used in modest applications, it is not clear how they relate or how to combine them to solve large-scale problems. This knowledge gap manifests itself by the lack of a unified view across all model forms, which leaves many questions unanswered. For example, how are constrained MDPs (CMDPs) related to options and can they be combined? What does it mean to perform an action if it can induce updates in multiple models? How can multiple models transfer control to one another if their state and/or action spaces are different? More generally, how can we create a principled mathematical framework that enables the integration of multiple models with these concepts?

We propose the notion of a policy network that helps us begin to answer these questions. It is a graph in which the vertices denote a set of policies and the edges denote their dependencies. A set of policies associated with a vertex refers to a state and action space that can be shared or distinct from any other vertices. A policy constraint edge enforces a restriction on a vertex's set of policies from another vertex. A policy transition edge defines a state transition in a vertex's state space or a transfer of control to another vertex. The objective is to maximize expected reward in the induced hierarchy of constrained semi-Markov decision processes following the graph's dependency structure.

Our primary contributions also form the sections of the paper: (1) a formal definition of policy networks and their properties; (2) a theoretical analysis that proves their generality, encapsulating prior models such as CMDPs and options; and (3) an implementation of the approach in a home healthcare robot acting in a real household environment.

Background

A *multi-objective Markov decision process (MOMDP)* is defined by $\langle S, A, T, \mathbf{R} \rangle$. S is a set of states. A is a set of actions. $T: S \times A \times S \to [0,1]$ is a state transition such that $T(s,a,s') = Pr(s'|s,a)$. $\mathbf{R} = [R_0, \ldots, R_k]^T$ is a vector of reward functions such that $R_i: S \times A \to \mathbb{R}$. A *policy* may be stochastic $\pi: S \times A \to [0,1]$ or deterministic $\pi: S \to A$. Π refers to any such set of policies. For a stochastic policy π, infinite horizon objectives, with discount factor $\gamma \in [0,1)$, have a *value* $\mathbf{V}^\pi: S \to \mathbb{R}^{k+1}$ for state s defined as:

$$\mathbf{V}^\pi(s) = \sum_{a \in A} \pi(s,a) \Big(\mathbf{R}(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \mathbf{V}(s') \Big).$$

A *constrained MDP (CMDP)* (Altman 1999) is a MOMDP with the objective:

$$\begin{array}{ll} \text{maximize} & V_0^\pi(s^0) \\ \text{subject to} & -V_i^\pi(s^0) \leq c_i, \quad \forall i \in \{1, \ldots, k\} \end{array} \tag{1}$$

and given constraints $c_i \in \mathbb{R}$. Rewards are commonly treated as costs $R_i = -C_i$ for each V_i. An *optimal* policy π^* obtains a maximal value $V_0^*(s^0)$.

A *semi-Markov decision process (SMDP)* (Puterman 1994) is an MDP in which control of the system is sojourn, relinquished to a so-called *natural process*, defining distinct *decision epochs* each with a *sojourn time*—duration the agent was not in control. The discrete time SMDP may be defined by $\langle S, A, T, F, R, \rho \rangle$. $T: S \times A \times \mathbb{N} \times S \to [0,1]$ is a state transition that includes the sojourn time τ such that $T(s,a,\tau,s') = Pr(s'|s,a,\tau)$. $F: S \times A \times \mathbb{N} \to [0,1]$ is the probability mass function (PMF) for the sojourn time τ such that $F(s,a,\tau) = Pr(\tau|s,a)$. $\rho: S \times A \times \mathbb{N} \to \mathbb{R}$ is the expected reward rate for the sojourn time. For a deterministic policy π, the *value* $V^\pi: S \to \mathbb{R}$ is defined as:

$$V^\pi(s) = \mathcal{R}(s, \pi(s)) + \sum_{s' \in S} \sum_{\tau=1}^{\infty} \gamma^\tau Pr(\tau, s'|s, \pi(s)) V^\pi(s'),$$

with state transition $Pr(\tau, s'|s,a)$ and:

$$\mathcal{R}(s,a) = R(s,a) + \sum_{\tau'=1}^{\infty} F(s,a,\tau') \sum_{\tau=1}^{\tau'-1} \rho(s,a,\tau).$$

Notationally, continuous time SMDPs define F as the cumulative distribution function (CDF) to facilitate its integral and derivative equations, whereas we equivalently define F as its PMF to facilitate discrete time equations (Howard 1971).

The options framework (Sutton, Precup, and Singh 1999) add special actions to an MDP, called options, that execute a complete policy, performing the actions of the agent until it stochastically returns control. It forms a special type of discrete time SDMP defined by $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_k\}$ with $\mathcal{O}_i = \langle \mathcal{I}_i, \pi_i, \beta_i \rangle$. For *Markov options*, $\mathcal{I}_i \subseteq S$ is a set of admissible initiation states, $\pi_i: S \to A$ is a policy, and $\beta_i: S \to [0,1]$ is the probability of terminating the option at each state. *Semi-Markov options* are instead defined over histories $\bar{H}$—sequences $\bar{h} = \langle s^0, a^0, s^1, a^1, \ldots \rangle$—with $\pi_i: \bar{H} \to A$ and $\beta_i: \bar{H} \to [0,1]$.

An *infinite horizon MDP* is generalized by both: (1) a CMDP with no constraints ($k=0$), and (2) a discrete time SMDP that immediately returns control ($F(s,a,1)=1$).

Policy Networks

In general, policy networks are graphs in which vertices denote sets of policies for a reward function and edges denote policy dependences among them. The objective is to capture the relations among distinct decision-making components to solve large multi-objective hierarchical problems.

Specifically, a **policy network** is a sequential decision-making model defined by a directed graph $\langle V, E \rangle$:

- V is a finite set of vertices such that each $v \in V$ denotes a set of policies Π_v for reward $R_v: S_v \times A_v \to \mathbb{R}$; and
- E is a finite set of edges such that each $\langle v, w \rangle = e \in E$ is:
  - policy constraint Π_e enforces $\Pi_w \subseteq \Pi_e$; and/or
  - policy transition T_e defines $T_e: S_v \times A_v \times S_w \to [0,1]$ as a partial function equal to $Pr(w, s'_w | s_v, a_v)$.

The execution of a policy network operates over discrete time steps $t \in \mathbb{N}$ as a form of *Markov reward process*. Each vertex v has a state space S_v and action space A_v for its policy and reward. Both, one, or neither space might be shared by any number of other vertices—that is, the same random variable in *dynamic Bayesian networks* or the same uncertainty or decision nodes *influence diagrams*. Each unique state space, say S_v for vertex v, has an initial state $s_v^0 \in S_v$.

As in (PO)MDPs, to **perform** an action is simply the act of conditioning on the action so as to induce an update in the underlying vertex v's stochastic process following the **state transition** distribution $Pr(w, s'_w | s_v, a_v)$. This probability distribution describes the state transition within the state space of v (i.e., $w = v$ and $s'_w = s'_v \in S_v$) as well as across other state spaces used by other vertices (i.e., $w \neq v$ and $s'_w \in S_w$). Policy networks require full specification of $Pr(w, s'_w | s_v, a_v)$, via the collection of functions T_e. In its simplest form, if v only transitions to itself by $T_e = T_{vv}$, then T_e is equivalent to a typical (PO)MDP state transition. Any induced state transition also induces a reward from R_v.

At each time step, any **controller** vertex v performs the action $\pi_v(s_v) \in A_v$ at their current state $s_v \in S_v$ from available policy $\pi_v \in \Pi_v$. Actions performed by v may result in a state transition to a different vertex w's state space. We call this a **transfer of control**, with the controller changing from v to w who now performs the action $\pi_w(s_w) \in A_w$ at their state $s_w \in S_w$ from available policy $\pi_w \in \Pi_w$. Each policy network has an initial controller $v^0 \in V$.

From an initial controller, we derive a graph describing the direct constraint or transfer dependencies among two vertices. Formally, the **dependency graph** $\langle V, E' \rangle$ is a directed acyclic graph (DAG) with $E' \subseteq E$ ensuring all paths from each vertex lead to the initial controller v^0. Generally, we consider any such DAG that contains a maximal number of edges E' from E. A common special case is simply the *shortest path tree*. Following the dependency graph, we can denote the **ancestors** of vertex v as $\mathcal{A}(v) \subset V$, its **parents** as $\mathcal{P}(v) \subseteq \mathcal{A}(v)$, its **descendants** as $\mathcal{D}(v) \subset V$, and its **children** as $\mathcal{C}(v) \subseteq \mathcal{D}(v)$.

Hierarchy of Constrained Semi-MDPs

A policy network induces a dependency graph that induces a hierarchy of constrained semi-Markov decision processes (CSMDP). Each CSMDP is recursively dependent, starting at the furthest vertices and ending at the initial controller.

Relative Times In discrete time SMDPs (Puterman 1994), there are three notions of time which policy networks share. These are relative to a vertex v, but subscripts will be omitted when it is not ambiguous. First, the **natural process time** is denoted by $\tau \in \mathbb{N}$. It refers to the total number of time steps following the state transitions for v's CSMDP. Second, a **decision epoch** is denoted by $t \in \mathbb{N}$. It refers to a time interval within the natural process time for v's CSMDP: $[\tau^1 + \cdots + \tau^t, \tau^1 + \cdots + \tau^{t+1})$. Third, a **sojourn time** is denoted by $\tau^t \in \mathbb{N}$ for decision epoch t. It refers to the duration of a decision epoch. This notation is overloaded, as it refers to both this duration and the random variable that determines this duration (F detailed below). In summary, vertex v's relative decision epochs are when it is a controller, and its relative sojourn times are the duration between being in control.

Relative State Let ancestors that share v's action space be $\overline{\mathcal{A}}(v) = \{w \in An(v) | A_w = A_v\}$. Only these ancestors can directly affect v's state, because if one were to become a controller then it would perform action, which by definition would induce state updates in v. Variables which have this consideration use the same notation, such as sojourn time $\overline{\tau}^t$.

Since each decision epoch's sojourn time reflects a vertex v's perspective on the effects ancestors have on its Markov reward process, its CSMDP omits the time spent by ancestors which do not share its action space. Consequently, the CSMDP state must consider which relevant ancestor is in control $w \in \overline{\mathcal{A}}(v)$, its state $s_w \in S_w$, and the vertex's own current successor state s'_v. Formally, this is defined by $\overline{s}_v = \langle w, s_w, s'_v \rangle$ with all such states denoted as $\overline{S}_v$. This space represents the CSMDP's states while the natural process—ancestors of v—is in control.

Relative State Transitions The relative CSMDP stochastic process follows $Pr(\overline{\tau}, \overline{s} | s_v, a_v)$. It refers to the probability that the next decision epoch for v occurs at or before sojourn time $\overline{\tau}$ and has successor state s'_v, with ancestor w's state s_w, after v had performed action a_v in state s_v.

Additionally, as in discrete time SMDPs, we may write $Pr(\overline{\tau}, \overline{s} | s_v, a_v) = F_v(s_v, a_v, \overline{\tau}) T_v(s_v, a_v, \overline{\tau}, \overline{s})$ given the sojourn time distribution following F_v and the state transition distribution following T_v. Note that as in general SMDPs, there are equivalent representations of $Pr(\overline{\tau}, \overline{s} | s_v, a_v)$ using different definitions of F_v and T_v, which may be used here as well if desired.

We can compute both $F_v(s_v, a_v, \overline{\tau})$ and $Pr(\overline{s} | s_v, a_v, \overline{\tau})$ directly by constructing a Markov chain $M^{|\overline{S}| \times |\overline{S}|}$ from the policy network and its dependency graph. The former is computed by marginalization over the Markov chain's transitions to states of the form $\langle v, s_v, s_v \rangle$, that is, transitioning back to v after sojourn time $\overline{\tau}$ from $M^{\overline{\tau}}$. The latter is computed directly following $M^{\overline{\tau}}$, as the $\overline{\tau}$-th iteration of the Markov chain determines these probabilities. There are two important properties regarding Markov chain M. First, as the state only needs to consider ancestors $\overline{\mathcal{A}}(v)$, all other ancestors $\mathcal{A}(v) - \overline{\mathcal{A}}(v)$ are summarized by their corresponding transfer control probabilities. Second, all policy transitions T_e outside of v's dependency graph are treated as absorbing states in v's CSMDP. This naturally captures the construction of abstractions with collections of subtasks: a subtask's goal or terminal states transfer control back to its parent after it completes the task it was designed to handle.

Relative Rewards The discrete time CSMDP has two rewards: (1) immediate reward R_v, and (2) expected reward gained at rate $\rho_v : S_v \times A_v \times \mathbb{N} \to \mathbb{R}$. Specifically, following SMDPs, $\rho_v(s_v, a_v, \overline{\tau})$ is defined as the reward rate at some sojourn time $\overline{\tau}$ after action a_v was performed in state s_v, but before the next action is performed. Thus, we have:

$$\rho_v(s_v, a_v, \overline{\tau}) = \gamma_v^{\overline{\tau}} \sum_{\overline{s} \in \overline{S}} R_v(s'_v, \pi_w(s_w)) Pr(\overline{s} | s_v, a_v, \overline{\tau}), \quad (2)$$

with a discount factor $\gamma_v \in [0, 1)$. The resulting reward is denoted $\mathcal{R}_v : S_v \times A_v \to \mathbb{R}$ and is written as:

$$\mathcal{R}_v(s_v, a_v) = R_v(s_v, a_v) + \sum_{\overline{\tau}'=1}^{\infty} F_v(s_v, a_v, \overline{\tau}') \sum_{\overline{\tau}=1}^{\overline{\tau}'-1} \rho_v(s_v, a_v, \overline{\tau}). \quad (3)$$

Relative Constraints As this is a *constrained* SMDP, the space of available policies *used to perform action* is restricted by policy constraint edges. Formally, each vertex v refers to a policy space such as $\Pi_v \subseteq \{\pi : S_v \to A_v\}$. This can be all possible policies or any non-empty subset, such as in the options framework or CMDPs. Options define $\Pi_v = \{\pi_v\}$ as a single fixed policy that *is* the option itself π_v (Sutton, Precup, and Singh 1999). CMDPs define $\Pi_v \subset \bigcap_i \Pi_{iv}$ by a restriction in this policy space by each other objective i's policy set Π_{iv} that satisfies constant c_i (Altman 1999).

In general, the set Π_v is defined iteratively, by computing the policy sets for all ancestors in $\mathcal{A}(v)$, ending with v's own edge constraint Π_{vv}, if any exists. Formally, Π_v ensures:

$$\pi_v \in \Pi_v \subseteq \Pi_{vv} \subseteq \bigcap_{w \in \mathcal{P}(v)} \Pi_{wv} \quad (4)$$

with v's final chosen policy being any $\pi_v \in \Pi_v$, for any Π_{vv} and Π_{wv} that exist in E.

To focus our discussion, we consider policy constraint edges Π_e in terms of a bound on regret from expected value (defined in the next section) up to a slack—allowable deviation from optimal. Formally, policy constraint edge $e = \langle v, w \rangle$ has a slack $\delta_{vw} \geq 0$ such that:

$$\Pi_{vw} = \{\pi \in \Pi_v | V_w^*(s_w^0) - V_w^\pi(s_w^0) \leq \delta_e\}. \quad (5)$$

While we focus on CMDPs in this paper, the lexicographic MDP (LMDP) (Wray, Zilberstein, and Mouaddib 2015), LPOMDP (Wray and Zilberstein 2015a), and possibly other forms of slack (Wray, Kumar, and Zilberstein 2018) can be generalized by policy networks as well. However, we leave this discussion to future work.

Objective Function

The **infinite horizon** objective of a policy network is defined recursively such that each vertex v's objective to find a policy $\pi_v \in \Pi_v$ that maximizes the expected reward starting from initial state s_v^0 subject to its ancestors:

$$\mathbb{E}\Big[\sum_{t=0}^{\infty}\gamma_v^t\mathcal{R}_v(s_v^t,\pi_v(s_v^t))\Big|\pi_v,s_v^0,\forall w\in\mathcal{A}(v),\pi_w,\Pi_w\Big] \qquad (6)$$

with s_v^t denoting the random variable for the state of v at its decision epoch t generated following the policy network's dependency graph state transitions $Pr(\overline{\tau},\overline{s}|s_v,a_v)$, and the stationary ancestor policies π_w and policy sets Π_w. To focus the discussion, we consider an infinite horizon; however, other objectives follow in the natural way.

For a policy $\pi_v \in \Pi_v$, the **value** $V_v^\pi : S_v \to \mathbb{R}$ is the expected reward at state s_v following the Bellman equation:

$$V_v^\pi(s_v) = \mathcal{R}_v(s_v,\pi_v(s_v)) + \sum_{\overline{s}\in\overline{S}}\sum_{\overline{\tau}=1}^{\infty}\gamma^{\overline{\tau}}Pr(\overline{\tau},\overline{s}|s_v,\pi_v(s_v))V_v^\pi(s_v'). \qquad (7)$$

A policy $\pi_v^* \in \Pi_v$ is **optimal** if it obtains the maximal value V_v^*. This optimal value can be computed by the Bellman optimality equation over each state s_v:

$$V_v^*(s_v) = \max_{a_v\in A_v}\Big(\mathcal{R}_v(s_v,a_v) + \sum_{\overline{s}\in\overline{S}}\sum_{\overline{\tau}=1}^{\infty}\gamma^{\overline{\tau}}Pr(\overline{\tau},\overline{s}|s_v,a_v)V_v^*(s_v')\Big). \qquad (8)$$

Stationarity

For a controller v and its policy set Π_v, the chosen policy $\pi_v^t \in \Pi_v$ for performing actions at time t can remain constant or vary over time. Formally, if $\pi_v^t = \pi_v$ for all time t, then we call the vertex's **policy stationary**, otherwise it is **non-stationary**. In the tradition of MDPs and planning, policies are commonly stationary. However, even solutions computed offline can have time-varying non-stationary policies. The holistic perspective policy networks affords a broader view of stationarity that includes to the behavior of *online* algorithms as well, which vary their policy over time in online planning (Ye et al. 2017) and reinforcement learning (Sutton and Barto 1998). The analysis is specific to the online scenario and therefore we leave this analysis to future work.

Since policy networks relate sets of policies to one another, the set of policies Π_v^t at a time t can also remain constant or vary over time. Formally, if $\Pi_v^t = \Pi_v$ for all time t, then we call the vertex's **policy set stationary**, otherwise it is **non-stationary**. In a simple MDP or POMDP within a policy network, the policy set trivially remains constant. However, in a growing number of online *models*—such as MODIA used in autonomous vehicles (Wray, Witwicki, and Zilberstein 2017)—the set of policies is constantly adjusted online. While this is easily described in a policy network, their formal analysis is nuanced and specific to the assumptions for each online scenario. For this reason, we leave any such extensive analysis to future work, favoring a description of one such online model in terms of policy networks to illustrate the approach.

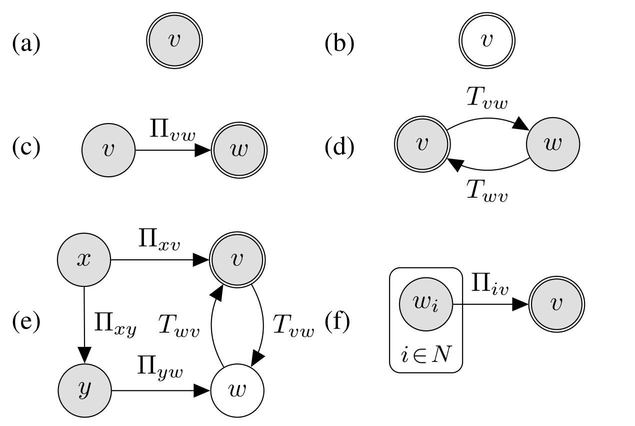


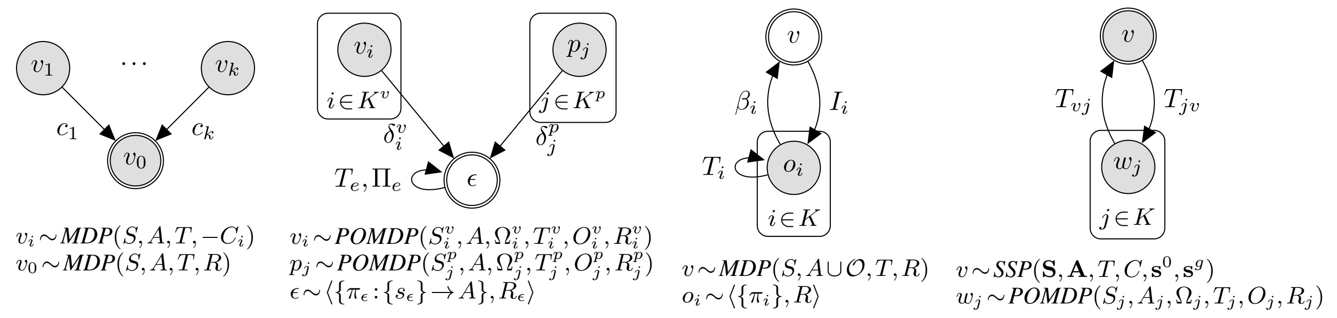
Figure 1: Basic examples of the graphical notation used to represent policy networks, with each $v \in V$ following some $v \sim \textit{MDP}(S_v, A_v, T_v, R_v)$: (a) stationary vertex; (b) non-stationary vertex; (c) constraint edge; (d) a transfer of control; (e) a mixture of these previous concepts; and (f) plate notation denoting a set of N constraints.

Graphical Representation

Policy networks continue the tradition set by probabilistic graphical models (PGMs) with a clean and powerful graphical representation. This allows for rich complex decision-making with many objectives and levels of abstraction to be easily described, analyzed, and implemented.

Figure 1 covers basic policy network notation. Each vertex as $v \in V$ is a circle and each directed edge $e \in E$ as an arrow. Edges are directed and denote their policy dependencies by any relevant variables. For example, policy constraints are denoted by their policy set Π_e and policy transitions are denoted by their function T_e. When it is not ambiguous, it is suffices to denote parameters instead, such as slack δ_e or an options' initiation set I_e or termination function β_e. Vertices and edges are lowercase; their sets are uppercase. The initial controllers v^0 are denoted by double-lined circles. Stationary vertices are filled-in—e.g., solved by *offline* algorithms. In contrast, non-stationary vertices which are *not* filled-in—e.g., solved by *online* algorithms. Plate notation may be used to easily group sets of similarly defined vertices.

Any vertex v which follows a standard MDP, POMDP, etc. model uses the notation $v \sim \textit{MDP}(\cdot)$, $v \sim \textit{POMDP}(\cdot)$, etc. in the tradition of PGMs. Intuitively, the "$\sim$" symbol refers to "selecting" a policy from a policy space. This notation is used for convenience. It completely describes the vertex's policy set Π, reward R, and an implicit self-loop edge with transition T and constraint Π^*—enforcing only optimal policies, as applicable. Formally, this extra notation means $v \sim \textit{MDP}(S, A, T, R)$ defines v with policies $\Pi \subseteq \{\pi : S \to A\}$ with reward $R : S \times A \to \mathbb{R}$. Additionally, it defines the *implicit edge*—that is, merely not graphically drawn—self-looping edge $e = \langle v, v\rangle \in E$ with policy constraint $\Pi \subseteq \Pi_e = \Pi^*$ and policy transition $T_e = T$. POMDPs are similarly defined as $v \sim \textit{POMDP}(S, A, \Omega, T, O, R)$ since they are a special form of continuous state MDP called a *belief MDP* (Kaelbling, Littman, and Cassandra 1998).



v_1
$\cdots$
v_k
c_1
c_k
v_0
$v_i \sim \mathit{MDP}(S,A,T,-C_i)$
$v_0 \sim \mathit{MDP}(S,A,T,R)$
v_i
$i \in K^v$
p_j
$j \in K^p$
δ_i^v
δ_j^p
T_e, Π_e
ϵ
$v_i \sim \mathit{POMDP}(S_i^v, A, \Omega_i^v, T_i^v, O_i^v, R_i^v)$
$p_j \sim \mathit{POMDP}(S_j^p, A, \Omega_j^p, T_j^p, O_j^p, R_j^p)$
$\epsilon \sim \langle\{\pi_\epsilon : \{s_\epsilon\} \to A\}, R_\epsilon\rangle$
v
β_i
I_i
T_i
o_i
$i \in K$
$v \sim \mathit{MDP}(S, A \cup \mathcal{O}, T, R)$
$o_i \sim \langle\{\pi_i\}, R\rangle$
v
T_{vj}
T_{jv}
w_j
$j \in K$
$v \sim \mathit{SSP}(\mathbf{S}, \mathbf{A}, T, C, \mathbf{s}^0, \mathbf{s}^g)$
$w_j \sim \mathit{POMDP}(S_j, A_j, \Omega_j, T_j, O_j, R_j)$

(a) CMDP (b) MODIA (c) Options (d) SAS

Figure 2: Four policy networks: (a) a constrained MDP; (b) MODIA; (c) the options framework, traditionally for a reinforcement learning agent; and (d) semi-autonomous systems, as a macro-action or subtask example.

Theoretical Analysis

We now show the generality of policy networks by proving that they can encapsulate various models such as the options framework and CMDPs. Additionally, this section also serves as a demonstration of the design of policy networks to provide guidance for how to create them. Proposition 1 begins with a simple but important statement regarding policy networks' generality beyond MDPs and POMDPs.

Proposition 1. *Policy networks generalize (PO)MDPs.*

Proof. For any MDP, we must construct an equivalent policy network. Let $V = \{v\}$ with $v \sim \langle\Pi, R\rangle$. Let $E = \{e = \langle v, v\rangle\}$ with transition $T_e = T$ and $\Pi_e = \{\pi | V^*(s^0) = V^\pi(s^0)\}$. Thus, $Pr(\overline{\tau}, \overline{s}|s, a) > 0$ only if $\overline{\tau} = 1$ and $Pr(1, \overline{s}|s, a) = T(s, a, s')$, also implying $\mathcal{R}(s, a) = R(s, a)$. Thus, Equation 7 becomes the MDP Bellman equation. □

Next, we consider two related cases of *policy constraint edges*: CMDPs and MODIA in Propositions 2 and 3, respectively. Here, constraints limit the space of policies from parent vertices to a child controller vertex. CMDPs represent a policy network with a shared both state and action space, constrained offline with stationary policies. MODIA represents a policy network with a different state space but a shared action space—illustrating how performing action can simultaneously affect many models—constrained online with non-stationary policy sets.

Proposition 2. *Policy networks generalize CMDPs.*

Proof. For any CMDP, we must construct an equivalent policy network. See Figure 2 (a). Let $V = \{v_0, \ldots, v_k\}$ with $v_0 \sim \mathit{MDP}(S, A, T, R_0)$, $v_i \sim \mathit{MDP}(S, A, T, -C_i)$, $v^0 = v_0$, and stochastic policies. Let $\{\langle v_i, v_0\rangle\} \subset E$ with $\Pi_{i0} = \{\pi | V_i^*(s^0) - V_i^\pi(s^0) \leq \delta_i\}$ and $\delta_i = V_i^*(s^0) + c_i$. Thus, the policy network has the same objective as the CMDP objective from Equation 1, and Equation 7 equal to the CMDP Bellman equation. □

MODIA (Wray, Witwicki, and Zilberstein 2017) is for autonomous vehicle (AV) decision-making about other entities: vehicles K^v and pedestrians K^p, analyzed a posteriori. Multiple POMDP models (i.e., v_i and p_j) describe each AV-entity pairwise interaction; each model is solved offline in isolate, resulting in stationary polices π_i^v and π_j^p. An *executor* $\epsilon : A^* \to A$ maps any tuple of action recommendations to a final action performed by the executor. This action updates the other models resulting in regret; e.g., for $i \in K^v$ regret is $r_i = V_i^*(b_i) - Q_i^*(b_i, \epsilon(\mathbf{a}))$. Following Wray *et al.* (2017), we consider risk-sensitive MODIA with LEAF, which assumes an ordering exists over actions $\succ$ in terms of safety. If a riskier action is performed $\pi_i(b_i) \succ \epsilon(\mathbf{a})$ then the model i experiences a regret lower bounded by $r_i \geq Q_i^*(b_i) - \underline{Q}$. A so-called LEAF executor that selects the safest action among the recommendations (i.e., $\forall i, \epsilon(\mathbf{a}) \succeq \pi_i(b_i)$ and $\exists j$ s.t. $\epsilon(\mathbf{a}) = \pi_j(b_j)$) minimizes the sum of one step regrets.

Proposition 3. *Policy networks generalize MODIA.*

Proof. For any MODIA, we must construct an equivalent policy network. See Figure 2 (b). Let $V = \{\epsilon\} \cup \{v_i\} \cup \{p_j\}$ with $v^0 = \epsilon$. Let each $v_i \sim \mathit{POMDP}(\cdot)$ and $p_j \sim \mathit{POMDP}(\cdot)$ as in the figure with shared A. Let $\epsilon \sim \langle\Pi_\epsilon, R_\epsilon\rangle$ for policies $\pi_\epsilon : S_\epsilon \to A$ with trivial state space $S_\epsilon = \{s_\epsilon\}$. Let $R_\epsilon(s_\epsilon, a)$ equal the index of a in the *reverse* of ordering $\succ$ over A. Let $\{\langle\epsilon, \epsilon\rangle\} \cup \{\langle v_i, \epsilon\rangle\} \cup \{\langle p_j, \epsilon\rangle\} \subset E$. Let $T_{\epsilon\epsilon}(s_\epsilon, \cdot, s_\epsilon) = 1$ be a self-loop transition and let $\Pi_{\epsilon\epsilon} = \{\pi | V_\epsilon^*(s_\epsilon) = V_\epsilon^\pi(s_\epsilon)\}$ select optimal executor policies. Let Π_ϵ be a non-stationary policy set that is defined by its parents' constraint edges $\Pi_{i\epsilon}^t = \{\pi : \{s_\epsilon\} \to A\} | V_i^*(b_i^t) - Q_i^*(b_i^t, \pi(s_\epsilon)) < \delta_i^v\}$, with a similarly defined $\Pi_{j\epsilon}$, that only allow executor actions with regret no greater than $\delta_i^v = V_i^*(b_i^t) - \underline{Q}$. By construction, the executor vertex ϵ's selected policy $\pi_\epsilon^* \in \Pi_\epsilon^t$ at time t maps its state s_ϵ to the safest action among recommendations. This is identical to the definition of LEAF, and thus minimizes the sum of one step regrets implicitly through constraints. □

Lastly, we consider two related cases of *policy transition edges*: the options framework and SAS in Propositions 4 and 5, respectively. Here, transfer of control happens between parent and child vertices, both online (options) and offline (SAS). Options represent a policy network with a shared state space and shared action space, learning online with a non-stationary policy. SAS represents a policy network with different state space and different action space—illustrating how how different models can interact—planning offline with a stationary policies.

Proposition 4. *Policy networks generalize options.*

Proof. For any set of options, we must construct an equivalent policy network. See Figure 2 (c). First, we consider Markov options. Let $V=\{v\}\cup\{o_i\}$ with $v^0=v$. Let $v\sim$ *MDP*$(S,A\cup\mathcal{O},T,R)$ and for each option $\mathcal{O}_i=\langle\mathcal{I}_i,\pi_i,\beta_i\rangle$ let $o_i\sim\langle\Pi_i,R\rangle$ with only the option policy $\Pi_i=\{\pi_i\}$. As v is traditionally a reinforcement learning agent, we simply represent the vertex's policy as non-stationary. Let R include the option reward $R(s,\mathcal{O}_i)=R(s,\pi_i(s))$. Let $\{\langle v,o_i\rangle\}\cup\{\langle o_i,v\rangle\}\cup\{\langle o_i,o_i\rangle\}\subset E$. Let $T_{vi}(s_v,\mathcal{O}_i,s'_i)=T(s_v,\pi_i(s_v),s'_i)[s_v\in\mathcal{I}_i]$ transfer control to the option, as allowed by $\mathcal{I}_i$, using Iverson bracket $[\cdot]$. Without loss of generality in MDPs, actions can be defined for each state $A(s_v)$, handling any invalid execution of options in states. Let $T_{iv}(s_i,a_i,s_v)=\beta_i(s_i)T(s_i,\pi_i(s_i),s_v)$ transfer control back to v stochastically following β_i; consequently, $T_i(s_i,a_i,s'_i)=(1-\beta_i(s_i))T(s_i,a_i,s'_i)$ accounts for β_i, too.

Semi-Markov options have a similar structure, except the shared state space is $\bar{H}$ instead of S, with $v\sim$ *MDP*$(\bar{H},A\cup\mathcal{O},T_v,R_v)$ and $o_i\sim\langle\Pi_i,R_v\rangle$. When v is in control, its transitions T remain at zero time states: $T_v(\bar{h}_v,a,\bar{h}'_v)=T(s_v,a,s'_v)[\bar{h}'=\langle s'_v\rangle]$ and $R_v(\bar{h},a)=R(s^t_v,a)$ for any $\bar{h}=\langle s^0_v,a^0,\ldots,a^{t-1},s^t_v\rangle$, again with $R_v(\bar{h},\mathcal{O}_i)=R(s^t_v,\pi_i(s^t_v))$. When options are in control, they transition over time by simply recording its action $\pi_i(s^t_i)$ and stochastic state s^t_i to iteratively construct a history embedded in its state space $\bar{H}$: $T_i(\bar{h}_i,a,\bar{h}'_i)=(1-\beta_i(s^t_i))T(s^t_i,a,s'_i)[\bar{h}'=\langle s^0_i,\ldots,s^{t-1}_i,a,s'_i\rangle]$. Lastly, transfer of control is similarly defined. Let $T_{vi}(\bar{h}_v,\mathcal{O}_i,\bar{h}'_i)=T(s^t_v,\pi_i(s^t_v),s'_i)[s^t_v\in\mathcal{I}_i]$ simply transfer to the option following $\mathcal{I}_i$. Let $T_{iv}(\bar{h}_i,a,\bar{h}_v)=\beta_i(s^t_i)T(s^t_i,a,s^0_v)$ transfer back to v following β_i, with it resetting time encoded in the state $\bar{h}_v$ for v.

In both cases, the value equation follows the option framework's discrete time SMDP Bellman equation. □

Semi-autonomous systems (SAS) (Wray, Pineda, and Zilberstein 2016) model the transfer of control of a single agent among a group of *actors* I that control it, such as transferring control between an AV and a human driver. It is built on a two-level hierarchy with a stochastic shortest path (SSP) problem (Bertsekas and Tsitsiklis 1991) reasons about transfer of control success and failure by executing a POMDP. The SAS state space $\mathbf{S}=S\times I$ includes the current actor, and the action space $\mathbf{A}=A\times I$ includes the desired next actor. The state transition $T:\mathbf{S}\times\mathbf{A}\times\mathbf{S}\to[0,1]$ follows the current actor's state transition $T_i:S\times A\times S\to[0,1]$. However, if a transfer is attempted at $\mathbf{s}=\langle s,i\rangle$, we multiply by $\rho:\mathbf{S}\times\mathcal{I}\times\mathcal{I}\to[0,1]$ as $\rho(\mathbf{s},\hat{i},i')=Pr(i'|\mathbf{s},\hat{i})$ denotes the probability that the next actor is i' given an attempt to transfer from i to $\hat{i}$. For each state-action pair, ρ is computed by a POMDP in a completely different state and action space that considers communication messages and belief about the state of the actor's preparedness to take control. Its execution ends in a collapsed absorbing belief state: success b^s, failure b^f, or abort b^a. A mapping $f:\{b^s,b^f,b^a\}\to I$ must be provided from these POMDP result states to the next SSP actor. Given its policy, the probability of reaching these three absorbing states is computed as ρ for use by the SSP.

Proposition 5. *Policy networks generalize SAS.*

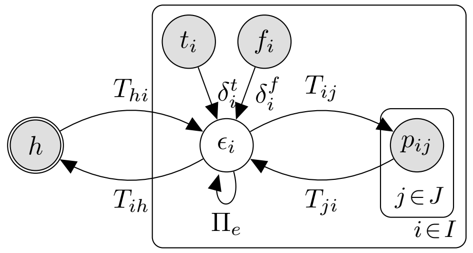
Proof. For any SAS, we must construct an equivalent policy network. See Figure 2. Let $V=\{v\}\cup\{w_j\}$ with $v\sim$ *SSP*$(\cdot)$ and $w_j\sim$ *POMDP*$(\cdot)$ as in the figure with distinct state and action. Without loss of generality and to remain inline with SAS, an SSP is used, which is akin to an MDP with discount $\gamma=1$, initial state $\mathbf{s}^0$, and goal state $\mathbf{s}^g$. For each SSP state-action pair there is a distinct POMDP w_j, with $K=\{j\in\mathbf{S}\times\mathbf{A}|j=\langle\mathbf{s},\mathbf{a}\rangle\wedge i\neq\hat{i}\}$. Let $\{\langle v,w_j\rangle\}\cup\{\langle w_j,v\rangle\}\subset E$. For T_{vj} and T_{jv}, we have the notation $j=\langle\mathbf{s},\mathbf{a}\rangle$, $\mathbf{s}=\langle s,i\rangle$, $\mathbf{a}=\langle a,\hat{i}\rangle$, and $\mathbf{s}'=\langle s',i'\rangle$. First, let $T_{vj}(\mathbf{s},\mathbf{a},b^0_j)=1$ be defined for any $i\neq\hat{i}$, always executing the corresponding POMDP starting at its b^0_j. Second, T_{jv} has three cases: for each $b\in\{b^s_j,b^f_j,b^a_j\}$ we have $T_{jv}(b,a_j,\mathbf{s}')=T_i(s,a,s')[f(b)=i']$. Since the action spaces are different ($\mathbf{A}\neq A_j$), v's CSMDP summarizes the state transitions of each w_j, resulting in v's Bellman equation producing a ρ for each w_j. This is identical to the SAS model. Thus, this policy network produces the same Bellman equations for the SAS's SSP and POMDPs. □

Evaluation: Home Healthcare Robot

Home healthcare robots serve in household and eldercare scenarios, providing solutions to a wide array of helpful tasks ranging from cleaning to medicine delivery (Robinson, MacDonald, and Broadbent 2014). Surveys conducted by Broadbent *et al.* (2009) analyzed and ranked the desired tasks the robot could do to help improve the lives of the elderly. Both elderly people and healthcare staff were surveyed. The top ranked needs include: (1) medicine notification and delivery; (2) forms of cleaning the house; and (3) monitoring, detecting, and helping with falls. We focus on a robot solution that captures all three.

The few mobile healthcare robots that exist tend towards hand-engineered decision-making systems that work well for their specific implementation (Graf, Hans, and Schraft 2004). One of the notable exceptions of a general model-based approach involves an early form of hierarchical POMDP (Pineau et al. 2003). They partitioned a single POMDP's action space into smaller groups, solving a collection of identical POMDPs with differing reduced action spaces. While successful, this early seminal work lacked the generality of a policy network—such handling multiple objectives in CMDPs, use different models (state and action spaces), or leverage grounding in SMDPs such as options.

Problem Description Consider a healthcare robot with a set of *high-level tasks* it must continuously complete. The *medicate task* is selected to complete by the high-level and requires navigating to the bathroom, retrieving medicine, finding the patient, and delivering it to them. The *clean task* is also selected and requires moving any out-of-place objects back in place while vacuuming. The *monitor task* must operate at all times, reactively interrupting any other task, and requires monitoring and detecting a fall of an elderly person. If confident in the detection, the robot should check on the person and autonomously call for the help of a healthcare professional. The *low-level path planning* must take special care to avoid obstacles to safely traverse the house.



$h \sim MDP(S^h, A^h, T^h, R^h)$
$t_i \sim POMDP(S_i^t, A_i^t, \Omega_i^t, T_i^t, O_i^t, R_i^t)$
$f_i \sim POMDP(S^f, A^f, \Omega^f, T^f, O^f, R^f)$
$\epsilon_i \sim \langle \{\pi_i^\epsilon : \{s_\epsilon, s_c\} \to A_i^t \cup A^f\}, R_i^\epsilon \rangle$
$p_{ij} \sim Harmonic(S^p, A^p, O^p, G_{ij}^p)$

Figure 3: The policy network for the home healthcare robot.

Name	V	$\|S\|$	$\|A\|$	$\|\Omega\|$	$\|\Gamma\|$	Time
High-Level Task Selector	h	16	4	—	—	<0.1
Medicate Task	t_1	289	13	5	1224	159.8
Clean Task	t_2	145	13	3	612	14.0
Fall Monitor/Assist Task	f_i	2	3	2	22	<0.1
Low-Level Path Planner	p_{ij}	17766	9	—	—	0.92

Table 1: The problem sizes and run times for each model.

Policy Network Solution Figure 3 shows the policy network for the healthcare robot. We provide a description of each vertex below. Also, Table 1 shows the problem sizes and results of solving these problems using *nova* (Wray and Zilberstein 2015b), with value iteration (VI), point-based VI (PBVI) (Pineau, Gordon, and Thrun 2006), and harmonic functions (Wray et al. 2016), for MDP, POMDP, and path planning models, respectively. PBVI has a policy size denoted as $|\Gamma|$. Harmonic function path planning ($Harmonic(\cdot)$ above) is equivalent to a special class of SSP with uniform state transitions, goals $G_{ij}^p \subset S^p$, and cost of 1 for obstacles $O^p \subset S^p$. All source code will be provided for complete problem descriptions and reproducability.

The high-level task h handles issues $I = \{t_1, t_2, f_i, \emptyset\}$ with $S^h = 2^I$ and $A^h = I$. Let $\emptyset$ denote a "complete" or "no-op" state and action here. The high-level h transfers control by T_{hi} to the start state of the corresponding task when selected as an action. Let a set of regions $\mathcal{R}$ (e.g., kitchen, bathroom, and bedroom) be given for the map. The medicate task t_1 has $S_1^t = \mathcal{R} \times \mathcal{R} \times \{Y, N\} \cup \{\emptyset\}$, denoting region locations for the robot and the person, as well as if the medicine is carried or not. $A_1^t = \mathcal{R}$ refer to navigation to a region by the path planner by T_{ij} and T_{ji}. $\Omega_2^t = \{Y, N\} \times \{Y, N\} \cup \{\emptyset\}$ refers to detection of a person or not, holding medicine or not, and completion. The clean task t_2 is similarly defined with $S_2^t = \mathcal{R} \times \mathcal{R} \cup \{\emptyset\}$ and $A_2 = \mathcal{R}$ as it searches for a location to clean. $\Omega_2^t = \{Y, N, \emptyset\}$ refers to detection of a person or not and completion. The monitor task f_i has state space $S^f = \{Y, N\}$ for if the person has fallen and needs help. $A^f = \{call, ask, \emptyset\}$ denotes calling for help, asking if the person needs help, and no-op. $\Omega^f = \{Y, N\}$ refers to detecting a fall or not. The executor ϵ follows as in a MODIA, with a preference for $call$ and ask over region navigation actions $\mathcal{R}$. T_{ih} transfers control back to h when in a task complete state s_c. The path planner p_{ij} navigates between regions $J = \mathcal{R} \times \mathcal{R}$ in the occupancy grid map in Figure 4.

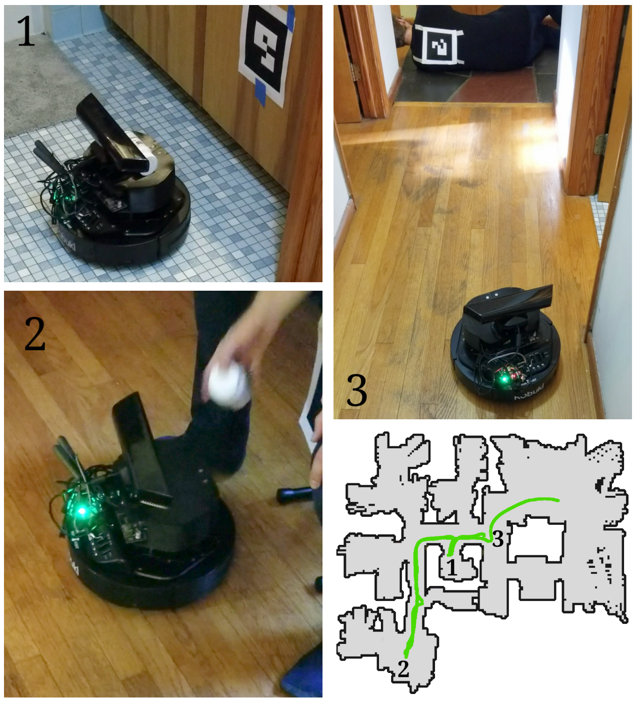


Figure 4: Experiments with the home healthcare robot using this policy network in the real household shown above. Three highlights are shown: (1) medicine retrieval for task t_1, (2) medicine delivery completion with transfer $t_1 \to h \to t_2$, and (3) interruption of cleaning task t_2 by detecting a fall with task f_i and calling for assistance.

Conclusion

We now revisit the questions posed in the introduction. First, how are CMDPs related to options and can these two models be combined? In policy networks, they are different types of edges between collections of distinct models: policy constraints and policy transitions. By simply adding any desired vertices and corresponding edges, we can easily combine both ideas, as evident by the previous section. Second, what does it mean to perform an action? In policy networks, it means conditioning on an action so as to induce a state update in any models that share the same action space. Third, how do these operate when the state and/or action spaces are different? Following the definition of performing an action, any shared action space induces state updates in the collection of models, as in options or more generally SMDPs. With different action spaces, the policies can still affect one another through transfer of control, treated as an abstraction or macro-action.

Finally, is there a principled mathematical model that enables the integrated design of multiple models with these concepts? Policy networks serve as an answer to this. They provide a theoretical model that generalizes select state-of-the-art models. The implementation shown here demonstrates they can successfully model and solve a challenging home healthcare robot domain. In summary, policy networks provide a general model for the reasoning component in real-world systems for long-term autonomy.

Acknowledgments

We thank the reviewers for their helpful comments. This research was supported by the National Science Foundation grant number IIS-1724101.

References

Altman, E. 1999. *Constrained Markov decision processes*. England: Chapman & Hall/CRC Press.

Bai, H.; Cai, S.; Ye, N.; Hsu, D.; and Lee, W. S. 2015. Intention-aware online POMDP planning for autonomous driving in a crowd. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 454–460.

Bertsekas, D. P., and Tsitsiklis, J. N. 1991. An analysis of stochastic shortest path problems. *Mathematics of Operations Research* 16(3):580–595.

Broadbent, E.; Tamagawa, R.; Kerse, N.; Knock, B.; Patience, A.; and MacDonald, B. 2009. Retirement home staff and residents' preferences for healthcare robots. In *Proceedings of the 18th IEEE International Symposium on Robot and Human Interactive Communication*, 645–650.

Graf, B.; Hans, M.; and Schraft, R. D. 2004. Care-o-bot ii—development of a next generation robotic home assistant. *Autonomous Robots* 16(2):193–205.

Howard, R. A. 1971. *Dynamic Probabilistic Systems: Semi-Markov and Decision Processes*. New York, NY: Wiley.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1):99–134.

Klein, L.; young Kwak, J.; Kavulya, G.; Jazizadeh, F.; Becerik-Gerber, B.; Varakantham, P.; and Tambe, M. 2012. Coordinating occupant behavior for building energy and comfort management using multi-agent systems. *Automation in Construction* 22:525–536.

Kochenderfer, M. J. 2015. *Decision Making Under Uncertainty: Theory and Application*. MIT Press.

Minsky, M. 1986. *The Society of Mind*. New York, NY: Simon and Schuster.

Pineau, J.; Montemerlo, M.; Pollack, M.; Roy, N.; and Thrun, S. 2003. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems* 42(3):271–281.

Pineau, J.; Gordon, G.; and Thrun, S. 2006. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research* 27:335–380.

Puterman, M. L. 1994. *Markov decision processes: Discrete stochastic dynamic programming*. New York, NY: John Wiley & Sons.

Robinson, H.; MacDonald, B.; and Broadbent, E. 2014. The role of healthcare robots for older people at home: A review. *International Journal of Social Robotics* 6(4):575–591.

Roijers, D. M.; Whiteson, S.; and Oliehoek, F. A. 2014. Linear support for multi-objective coordination graphs. In *Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems*, 1297–1304.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT Press.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2):181–211.

Tao, Y.; Wang, T.; Wei, H.; and Chen, D. 2009. A behavior control method based on hierarchical POMDP for intelligent wheelchair. In *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 893–898.

Wray, K. H., and Zilberstein, S. 2015a. Multi-objective POMDPs with lexicographic reward preferences. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 1719–1725.

Wray, K. H., and Zilberstein, S. 2015b. A parallel point-based POMDP algorithm leveraging GPUs. In *Proceedings of the AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*, 95–96.

Wray, K. H.; Ruiken, D.; Grupen, R. A.; and Zilberstein, S. 2016. Log-space harmonic function path planning. In *Proceedings of the 29th IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1511–1516.

Wray, K. H.; Kumar, A.; and Zilberstein, S. 2018. Integrated cooperation and competition in multi-agent decision-making. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 4751–4758.

Wray, K. H.; Pineda, L.; and Zilberstein, S. 2016. Hierarchical approach to transfer of control in semi-autonomous systems. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 517–523.

Wray, K. H.; Witwicki, S. J.; and Zilberstein, S. 2017. Online decision-making for scalable autonomous systems. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, 4768–4774.

Wray, K. H.; Zilberstein, S.; and Mouaddib, A.-I. 2015. Multi-objective MDPs with conditional lexicographic reward preferences. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 3418–3424.

Ye, N.; Somani, A.; Hsu, D.; and Lee, W. S. 2017. DESPOT: Online POMDP planning with regularization. *Journal of Artificial Intelligence Research* 58:231–266.

Reasoning and Learning in Real-World Systems for Long-Term Autonomy (LTA)
Papers from the AAAI 2018 Fall Symposium

Big Data and Deep Learning Models for Automatic Dependent Surveillance Broadcast (ADS-B)

Ying Zhao
Naval Postgraduate School
yzhao@nps.edu

Richard Wu
UMass Dartmouth
rwu@umassd.edu

Andrew Polk
UC Santa Barbara
polk@umail.ucsb.edu

Matthew Xi
Indiana University Bloomington
mxi@iu.edu

Tony Kendall
Naval Postgraduate School
wakendal@nps.edu

Abstract

ADS-B functions with satellite (GPS) rather than radar technology to more accurately observe and track air traffic. Aircraft equipped with an ADS-B Out transmitter sends position, altitude, heading, ground speed, vertical speed, call sign, and other aircraft information to a network of ground stations that relays the information to air traffic controllers and other aircraft. Our work in progress applies various big data tools and deep learning models such as convolutional neural networks to process and use the ADS-B big data to predict if a flight is commercial or military.

Introduction

ADS-B functions with satellite (GPS) rather than radar technology to more accurately observe and track air traffic. Aircraft equipped with an ADS-B Out transmitter sends their position, altitude, heading, ground speed, vertical speed, call sign, and other aircraft information to ground stations that relays the information to air traffic controllers and those who have ground ADS-B receivers. Pilots of aircraft equipped with a receiver for optional ADS-B receive traffic and meteorological information. Aircraft operating in most controlled U.S. airspace must be equipped for ADS-B Out by January 1, 2020 (FAA 2018).

With ADS-B operational across the country, pilots in equipped aircraft have access to air traffic services that provide a new level of safety, better situational awareness, and more efficient search and rescue.

Data Description

An aircraft can be identified by radar transponder identification, friend or foe (IFF) modes such as one, three, and five (military). This is done by Line of Sight (LOS) air traffic control ground radar stations. For improved (cooperative) surveillance for flight separation and control an aircraft can have an Automatic Dependent Surveillance-Broadcast (ADS-B) Out system to broadcast its identification and location from the aircrafts GPS to LOS receiving ground stations and other aircraft (around a 150 mile range). According to International Civil Aviation Organization (ICAO), a.k.a, the international "FAA", states that notable outcomes of using ADS-B include a new frequency allocation for space-based ADS-B reception, enabling tracking of aircraft globally including remote and polar regions. Plans are to fully employ ADS-B and the related infrastructure by 2020 (FAA 2018).

Table 1: 3-month ADS-B Data Divided into 7 Data Sets

Name	Month of Data
Data set 7_1	July, 2016,Train
Data set 7_0	July, 2016,Test
Data set 7_2	July, 2016,Test
Data set 8_0	August, 2016,Test
Data set 8_1	August, 2016,Test
Data set 6_0	June, 2016,Test
Data set 6_1	June, 2016,Test

Four terabytes ADS-B data (ADSBexchange.com 2017), were downloaded from the website ADS-B Exchange sampled every minute, for the whole year (6/2016 to 6/2017) and processed the selected fields and geo-spatial areas for a three month time period using the NPS high performance computer. The data are divided into 7 data sets representing about 10 days of the flights for a selected region as shown Table 1. There are approximately 150,000 flights (tracks) in each data set. Our goal is to build ML/AI models such as lexical link analysis (LLA) and reinforcement learning (RL) algorithms to rapidly and accurately classify military and commercial aircraft based on kinematic characteristics. The objective of this paper is to investigate how the track data could be represented as images and fed into the deep learning algorithm such as convolutional neural networks (CNN) for high classification accuracy.

There are two versions of all the three data sets

- Data: Original kinematic attributes with numeric values for a whole track as follows:
  - Mean altitude (barometric)
  - Mean altitude change
  - Mean altitude change absolute
  - Mean speed
  - Mean speed change
  - Mean speed change absolute
  - Mean heading change
  - Mean heading change absolute
  - Total altitude change

  - Total altitude change absolute
  - Total heading change
  - Total heading change absolute
  - Total speed change
  - Total speed change absolute
  - Total duration

  The target attribute to predict is mil_true: if an airborne object is military (true) or commercial (false). The percentage of military flights is about 2%.

- Unique data: Discretized original kinematic at-tributes. For example, the "average altitude attribute is turned into the following categorical attributes:
  - Likelihood to be commercial when average altitude between 16,607 feet and 29,525 feet
  - Likelihood to be military when average altitude between 16,607 feet and 29,525 feet
  - Likelihood to be commercial when average altitude between 3,689 feet and 16,607 feet
  - Likelihood to be military when average altitude between 3,689 feet and 16,607 feet
  - Likelihood to be commercial when average altitude less than 3,689 feet
  - Likelihood to be military when average altitude less than 3,689 feet
  - Likelihood to be commercial when average altitude more than 29,525 feet
  - Likelihood to be military when average altitude more than 29,525 feet

  The values of discretization are computed automatically by examining the means and standard deviations of the original attributes. After discretization, the percentage of military flights is about 6%.

Methods

Data Exploration

Tableau (tab) can filter the data with any of the metrics included in the data set. Changing the sizes of the nodes based on the number of virtual tracks that pass through that point allows for the visualization of heavy traffic paths over a map. The example below clearly shows the most popular flight paths based on the darker lines on the map. Tableau also allows for coloring the points on the map though other parameters in the data. Tableau can display virtual tracks and statistics. Figure 1 shows examples of virtual tracks of aircraft for a selected region when the altitudes are higher than 18,000 feet for the training set 7_1. Figure 2 shows the virtual tracks when the tracks altitudes are less than 18,000 feet and total heading change for tracks are larger than 2000. Heading is the compass direction toward which an airborne object should be moving. The blue tracks are commercial and the orange tracks are military. Comparing Figure 1 and Figure 2, we can see that the kinematic characteristics for Figure 2 (i.e., flying below 18,000 feet and total heading change greater than 2000) are more likely to indicate military aircraft, while the kinematic characteristic (i.e., flying higher than 18,000 feet) for Figure 1 is more likely to indicate commercial aircraft.

Supervised Learning Using Weka

We also reviewed and tested supervised learning methods such as decision trees, logistic regression, nave Bayes, nearest neighbors, and random forest models in the open source data mining and machine learning tool Weka (Hall et al. 2009) as shown in Figure 3. K-nearest neighbors and random forest perform the best for both versions of the data. The part of the research gave us the baselines for comparison for the data set. Since ADS-B data are mostly commercial flights, even 2% flights are self-reported as military, these aircraft mostly fly similar to commercial aircraft based on their kinematic characteristics. This is the key reason for the difficulty of the classification task of this data set.

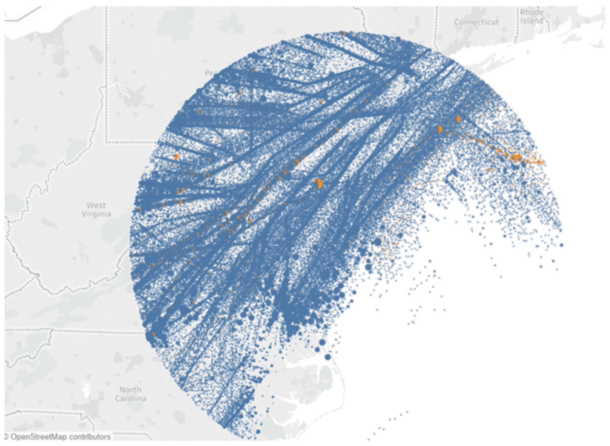


Figure 1: Sample virtual commercial tracks with altitude greater 18,000 feet

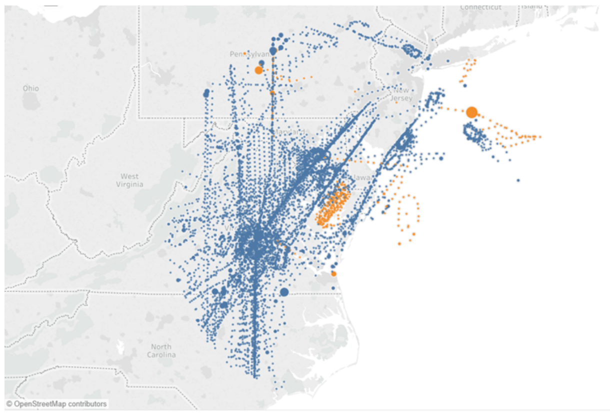


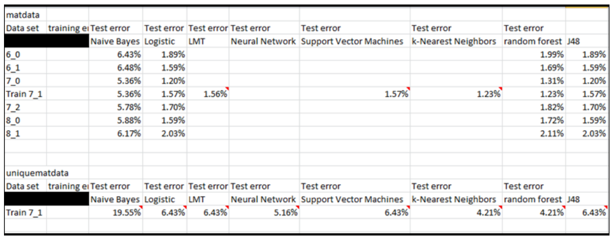
Figure 2: Map with altitude below 18,000 feet and total heading change greater than 2000. The blue tracks are commercial and the orange tracks are military

Convolutional Neural Networks (CNN)

CNN can train themselves to classify objects in images with high accuracy (CNN 2018). We explored if CNN can be used to predict military and commercial flights from the ADS-B track data. We represented the ADS-B track data as images and then apply CNN. The ADS-B data set contains a flights information such as the aircraft name, timestamp, speed, heading, altitude, military or commercial. We define a track to be the period between takeoff and landing. For each track, we focus on three in-put attributes i.e. speed, heading, and altitude and one output attribute i.e. its classification as military or commercial. Each track updates its kinematic data at one-minute time intervals.

For this particular data set, the color scale will represent altitude values ranging between -16,935 and 126,400. The negative altitudes are probably data errors. Figure 5 shows an example of a track heat map using the color scale in Figure 4 and altitude is the color or heat.

Segmenting Data into Tracks Microsoft Excel was used to open the data set as a .csv file. A python script was created to split the ADS-B data into tracks, or flight paths, and two data folders. One folder for all the civilian aircraft and one for military. The complete data was presorted by time and by aircraft ID. The python script split the full data set into smaller ones based on aircraft ID and then again based on the time between measurements. If the time between sensor measurements for the aircraft was greater than one hour a new track was formed. Afterward, another python program is written to plot each flight into one figure. The main problem with the flight heading data was the heading value was absolute and not relative causing the data to jump when



matdata									
Data set	training e	Test error	Test error	Test error	Test error	Test error	Test error	Test error	
[redacted]	[redacted]	Naive Bayes	Logistic	LMT	Neural Network	Support Vector Machines	k-Nearest Neighbors	random forest	J48
6_0		6.43%	1.89%					1.99%	1.89%
6_1		6.48%	1.59%					1.69%	1.59%
7_0		5.36%	1.20%					1.31%	1.20%
Train 7_1		5.36%	1.57%	1.56%		1.57%	1.23%	1.23%	1.57%
7_2		5.78%	1.70%					1.82%	1.70%
8_0		5.88%	1.59%					1.72%	1.59%
8_1		6.17%	2.03%					2.11%	2.03%
uniquematdata									
Data set	training e	Test error	Test error	Test error	Test error	Test error	Test error	Test error	
[redacted]	[redacted]	Naive Bayes	Logistic	LMT	Neural Network	Support Vector Machines	k-Nearest Neighbors	random forest	J48
Train 7_1		19.55%	6.43%	6.43%	5.16%	6.43%	4.21%	4.21%	6.43%

Figure 3: Supervised learning methods and error rates used for baselines and comparison

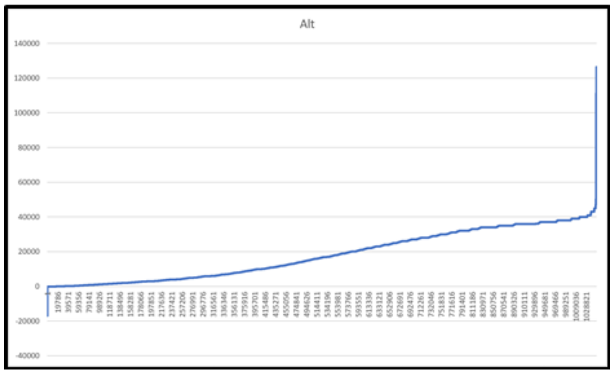


Figure 4: A graph showing the minimum and maximum altitude range over 10 days of ADS-B data from July 10th, 2016 to July 20th, 2016

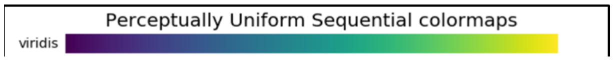


Figure 5: Sequential Color scale representing low altitudes on the left (purple) and high altitudes on the right (yellow)

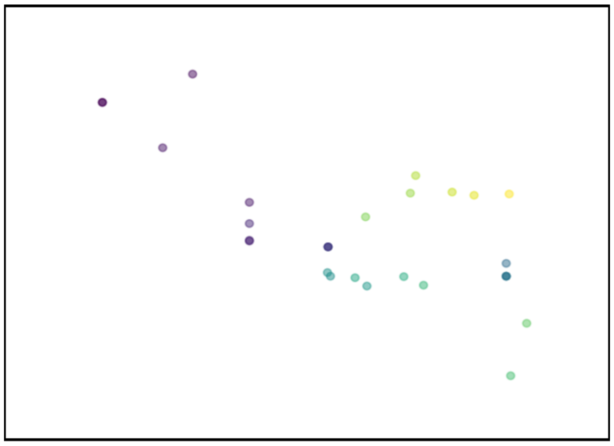


Figure 6: Heat map of speed (x-axis), heading (y-axis) and altitude (color) for one track

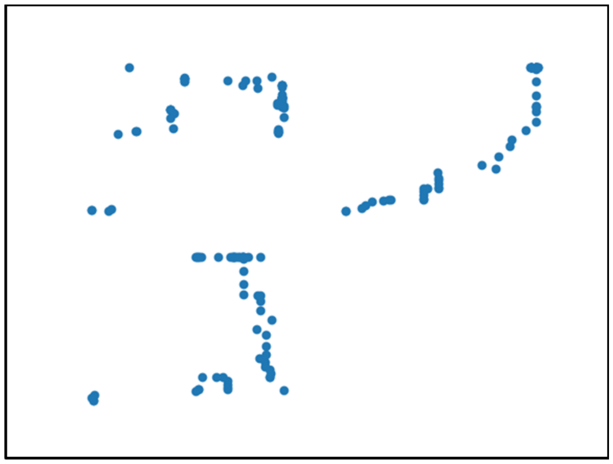


Figure 7: Three subplots demonstrating three different relations between speed, heading, and altitude

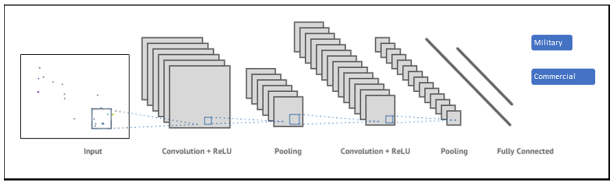


Figure 8: The CNN Architecture demonstrating a scatter plot to be classified as military or commercial

crossing the 0°or 180°. To smooth the track headings in the data was modified to fit between 90°and 90°. A full rotation of the heading represented this way is visually equivalent to that of a sine wave. Some data is lost in this representation as there is no way to discern 0 °from 180 but the relative change in heading between data points is saved. This smoothing method solves the problem of data jumping around when the heading changes from 1°to 359°as this will be shown as a change of 2°rather than 358°allowing for a cleaner visualization of the data for processing.

Transform Tracks into Images Before using a CNN, first we converted a whole track into an image. We represent the input attributes for a track as an image by plotting their relationship i.e. speed vs heading vs altitude as a color scale as shown in Figure 5 The color bar and axis of the graph have been turned off to reduce unnecessary noise in the image. A perceptually uniform sequential color map (Colormaps 2018) has been chosen to represent the altitude data. Sequential color maps change in lightness and saturation incrementally, using a single hue, and is suitable for information that has ordering as shown in Figure 4.

Figure 4 is a graph showing the minimum and maximum altitude for the train data set which is used to the color scale in Figure 4. The color scale represents the altitudes within the entire ADS-B data set, not within a single flight path. This is so that when comparing two flight paths (i.e. two plotted images) the color at a specific altitude would be represented as the same color.

In addition to the heat plot, we also tried to represent three pairwise (speed vs heading, heading vs altitude, altitude vs speed) kinematic feature scatter plots into one figure as another way to transform track data into images as shown in Figure 7. The top right section displays speed vs heading. The top left displays speed vs altitude, and the bottom left displays heading vs altitude. The borders of the subplot have been turned off to reduce the noise in the image.

Running CNN

For both methods, a whole track as a graph is re-sized from a 640x480 image to an 80x60 image. The sample ADS-B data for a testing CNN run contains 50,293 images of commercial flights and 2,316 images or 4.4% of military flights. The images were prepared to be fed into the CNN to classify commercial and military flights as shown in Figure 8.

The CNN in TensorFlow (Keras) performed ten iterations of training on both transformed images to produce an accuracy of 95.6% for classifying commercial flights.

We explored various big data and deep learning methods to classify ADS-B data sets. Supervised learning methods show certain ML accuracy. We also explored how to represent track data as images and then applied CNN for classifications. Our work in progress did provide promise that various big data tools and deep learning models such as convolutional neural networks can process and use the ADS-B big data to predict if a flight is commercial or military. Other software tools such as Tableau, Weka and even MS Excel show the potential that ADS-B data could be manipulated to identify commercial and military behavior to predict type of aircraft.

Conclusion and Future Work

The result of CNN remains a work in progress. Future work is needed, for example, using more data from different sources to enrich military aircraft. Also while our tools demonstrated some promising results we need to explore tools that handle big data and near real time data such as BDP (Big Data Platform) which is being used in related work.

Acknowledgements

Authors would like to thank the Naval Research Program at the Naval Postgraduate School and the Naval Research Enterprise Internship Program at the Office of Naval Re-search for the research support. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the U.S. Government.

References

ADSBexchange.com, L. 2017. Ads-b exchange.

CNN. 2018. Convolutional neural network, retrieved from http://cs231n.github.io/convolutional-networks/.

Colormaps. 2018. Colormaps,retrieved from https://matplotlib.org/tutorials/colors/colormaps.htmlsequential.

FAA. 2018. FAA,retrieved from https://www.faa.gov/nextgen/how_nextgen_works/new_technology/.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations* 11(1):10–18.

Tableau, https://www.tableau.com/.