Centralized learning and planning : for cognitive robots operating in human domains

Centralized Learning And Planning

For Cognitive Robots Operating In Human Domains

The research leading to this dissertation has received financial support from the RoboEarth project funded by the European Union Seventh Framework Program FP7/2007-2013 under grant agreement number 248942.

The research leading to this dissertation is supported by the research program of the Dutch Institute for Systems and Control (DISC). The author has successfully completed the educational DISC Graduate Program.

A catalogue record of this dissertation is available from the Eindhoven University of Technology Library under ISBN: 978-90-386-3601-6

Centralized Learning And Planning For Cognitive Robots Operating In Human Domains
Eindhoven: Technische Universiteit Eindhoven, 2014 – Proefschrift.

This thesis was prepared with the pdfLaTeX documentation system.
Cover Design: Gijs Hermans
Reproduction: Ipskamp Drukkers B.V., Enschede, The Netherlands

Centralized Learning And Planning

For Cognitive Robots Operating In Human Domains

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de rector magnificus prof.dr.ir. C.J. van Duijn,
voor een commissie aangewezen door het College voor Promoties,
in het openbaar te verdedigen
op dinsdag 15 april 2014 om 16.00 uur

door

Rob Josephus Maria Janssen

geboren te Eindhoven

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. L.P.H. de Goey
promotor:	prof.dr.ir. M. Steinbuch
copromotor:	dr.ir. M.J.G. van de Molengraft
leden:	prof.dr.ir. L.M.G. Feijs prof.dr. R. D'Andrea prof.dr.ir. S. Stramigioli prof.dr. H. Bruyninckx

Summary

Centralized Learning And Planning For Cognitive Robots Operating In Human Domains

As a technological answer to societal and economic problems that arise because of our aging society, advanced robotic systems are currently being developed that support health-care operatives in their daily occupations. Key ingredients that allow these robots to be successfully deployed for these activities are the ability to recognize their environment, to learn from experience and to decide upon a right course of actions.

Until now, the design approaches that allowed these robots to exploit these methods, focused primarily on the development of standalone robot applications. Autonomous robots, operating and learning individually, running their own computational algorithms, and reasoning about the world within their own small set of beliefs and assumptions.

With the rise of high-bandwidth and big-data processing mechanisms, it is now however possible for robots to share this information in runtime, similar to how humans share their knowledge over the Internet. This allows robots to communicate their knowledge to other robots instantaneously, and allows efficient storage and redistribution of learned concepts and task information on a global level.

Having this knowledge available in unified storage locations, forwards the idea of *centralized learning and planning*; one central system, that receives robot knowledge on a global level, stores learned concepts, and controls thousands of robots in a ubiquitous and highly optimized manner. An additional advantage of this type of centralized system, is that heavy computations required for the processing of data can be run here, requiring robots to only run small hardware control modules and lightweight client interfaces.

This thesis investigates the required components that are involved for such a system, with an emphasis on the required cognitive, learning, planning and execution components. For its main communication framework, a system is adopted that allows the deployment of computing environments and robot interfaces in a secure and straightforward manner. Additionally, web based service technologies are investigated as foundational methods for planning and execution, as their representational languages can be considered highly suitable for task related knowledge engineering in the robotics domain.

One of the system's integral components that will be discussed in this thesis describes a method for segmentation and tracking, that enables the cognition of maneuvering objects with movement trajectories, such as those typically encountered in human domains. Additionally, methods for learning will be investigated, as they enable the system to improve upon its existing knowledge and decision making capabilities. Both of these topics will be verified and tested in isolated experiments performed on the Eindhoven University automated football table. This small scale test environment provides detailed insights to the adopted methods, and allows identification of both strengths and weaknesses in well-controlled experiments.

Subsequently, methods for logic-based task planning and task execution will be discussed, as these methods are required for the systems deliberative and operational capabilities. Here, experiments will be conducted with a human-sized robot, performing tasks in a real household environment.

Further investigations will extend in this domain, where planning, execution and computational algorithms are now deployed on a cloud based computing platform. As this experiment will involve an additional robot with only minor computing capabilities, this chapter will show the importance of computational offloading for robots, and how tasks with multiple robots can be performed efficiently through centralized planning.

Samenvatting

Gecentraliseerd Leren En Plannen Voor Cognitieve Robots Opererende In Menselijke Omgevingen

Om het hedendaags probleem van onze verouderende samenleving door middel van technologie te kunnen verkleinen, worden momenteel geavanceerde robots ontwikkeld die de zorgsector kunnen bijstaan in hun dagelijkse taken. De belangrijkste ingrediënten die deze robots in staat stellen hun taken succesvol uit te voeren, bestaan uit het vermogen van deze robots om hun omgeving waar te kunnen nemen, om van hun ervaringen te kunnen leren, en om in staat te zijn een logische manier van handelen te kunnen bepalen.

Tot op heden, richtte men zich voor het toepassen van deze methoden voornamelijk op autonome applicaties. Volledig zelfstandige systemen, die individueel opereren en leren, hun eigen berekeningen uitvoeren, gebaseerd op hun eigen kleine voorstelling van de wereld om zich heen.

Met de komst van snelle en geavanceerde informatie verwerkingscentra, is het nu echter mogelijk voor robots om hun kennis direct met elkaar te delen, identiek aan hoe mensen hedendaags informatie met elkaar delen via het Internet. Dit stelt robots in staat om hun kennis instantaan met elkaar te delen, en maakt het mogelijk om over de gehele wereld geleerde concepten en taak informatie, op een efficiënte manier te kunnen opslaan en hergebruiken.

Het ter beschikking hebben van deze kennis op één centrale locatie, suggereert mogelijkheden voor *gecentraliseerd leren en plannen*; één centraal systeem, dat wereldwijd robot kennis ontvangt, deze kennis opslaat, en hiermee duizenden robots op de meest optimale manier aanstuurt. Een bijkomend voordeel van zulks een systeem, is dat zware berekeningen ook hier kunnen worden uitgevoerd, waardoor robots zelf slechts minimale rekencapaciteit hoeven te hebben en hierdoor klein en compact kunnen blijven.

In dit proefschrift wordt onderzocht welke componenten benodigd zijn voor zulks een systeem, waarbij de nadruk wordt gelegd op de benodigde volg, leer, planning en uitvoerende componenten. Voor het communicatie systeem wordt een reeds bestaand systeem gebruikt, dat het mogelijk maakt om berekeningen en het maken van verbindingen op een veilige en eenvoudige manier te kunnen uitvoeren. Bij het ontwerp van de planning en

uitvoerende componenten, worden bovendien web gebaseerde technologieën gebruikt als bouwstenen , daar ze als uitermate geschikt kunnen worden beschouwd voor het opstellen en representeren van taak gerelateerde kennis in het gebied van de robotica.

Een van de integrale componenten in het systeem dat in dit proefschrift zal worden behandeld, is een volg methode, die het mogelijk maakt om bewegende objecten te volgen die zich voortbewegen op een manier zoals men vaak tegenkomt in menselijke omgevingen. Aansluitend, worden leer methoden onderzocht, daar deze het mogelijk maken voor het systeem om van zichzelf te leren, en hierdoor betere beslissingen te kunnen maken in de toekomst. Deze twee methoden zullen beide worden onderzocht en toegepast op de autonome voetbaltafel van de Technische Universiteit Eindhoven, daar deze opstelling zich uitstekend leent voor het uitvoeren van experimenten op een gecontroleerde en overzichtelijke manier.

Hierop volgend, worden methoden voor logisch gebaseerd plannen en uitvoeren bediscussieerd, daar deze methoden zijn benodigd om het systeem te kunnen laten redeneren en taken uit te kunnen laten voeren. In dit hoofdstuk, zullen experimenten worden uitgevoerd met een mensachtige robot, die taken uitvoert in een volledig mensachtige omgeving.

Verder onderzoek zal zich uitbreiden in deze omgeving, waar nu de planning en uitvoerende componenten zullen worden geplaats op een cloud gebaseerd rekenplatform. Daar in dit experiment een robot zal worden betrokken die slechts minimale rekencapaciteiten heeft, zal dit hoofdstuk aantonen hoe belangrijk het is voor robots on hun berekeningen elders uit te kunnen laten voeren, en hoe taken met meerdere robots efficiënt kunnen worden uitgevoerd door middel van een gecentraliseerde aansturing.

Contents

Chapter 1

Introduction

"The main objective of RoboEarth is to develop a system design capable of carrying out useful tasks autonomously, in circumstances that were not planned for explicitly at design time."

"RoboEarth - Connecting Robots Worldwide", 2009

1.1 Motivation

1.1.1 A desire for new assistive technologies

A recent report of the United Nations about the world's aging population [1], indicates that in the upcoming years the ratio between elderly and young people will significantly increase. This increased ratio will have profound implications on the world's societal and economic situation, as a comparatively smaller group of work-capable people has to take care of a relatively larger group of health-care requiring elderly. New technologies are therefore required, that support this sector in tasks that are considered involuntary but required, such as laundry disposal, medicine delivery or vacuum cleaning, see Figure 1.1.

(a) (b) (c)

Figure 1.1: Several examples of modern day health-care assistants. The Tug laundry disposal robot (a), the Hospi delivery bot (b) and the Roomba vacuum cleaning robot (c).

1.1.2 The challenges

Introducing robots into this human oriented domain, requires technological advancements in multiple areas of robotics research however. As opposed to a robot's typical factory floor habitat, human domains are far more complex. They contain unpredictable natural events, such as pouring rain and blinding sun, but also the unpredictability of human activity itself, which may or may not follow from rational objective. Human domains also contain unstructured, multiform objects that require complex perception models to be perceived, and advanced modeling techniques to be tracked over time. Furthermore, human domains are ergonomically fully adapted to an average sized human; desiring robots to perform humans tasks, such as driving cars, walking stairs or even manipulating small articulated objects are big challenges for currently available platforms. Furthermore, the spoken and written language that humans use for describing concepts and to communicate on the performance of activities, often contains ambiguities and a lack of clear semantics [118]. These aspects make it difficult for robots to understand human concepts and their correct interpretations.

1.1.3 Robot design advancements

Allowing factory floor robots to handle these challenges therefore requires a new breed of robot; full human-sized mobile manipulators, with advanced cognitive and manipulation capabilities. The design of these autonomous systems is however a daunting task. First, an extensive amount of engineering needs to be spent on the development and integration of robust and suitable hardware components. Second, these robots needs to be equipped with advanced cognitive capabilities in order to understand the human domain, and to be able to decide upon a rational course of actions. Since the deployment of the first robot that was capable of rational reasoning (SRI's 'Shakey', 1966), a broad spectrum of cognitive robots has been developed. As a descendant of Shakey came SRI's 'Flakey' in 1984, which participated in the first AAAI robotics competition. From that point on, multiple systems were being developed in parallel, such as the University of Michigan's 'Carmel', Georgia Tech's 'Buzz' and IBM's 'TJ2', see Figure 1.2.

(a) (b) (c) (d) (e)

Figure 1.2: First breed of cognitive, self-deciding robots. SRI Shakey (a) and successor Flakey (b), University of Michigan Carmel (c), Georgia Tech Buzz (d) and IBM TJ2 (e).

Recent developments brought systems with advanced humanoid locomotion capabilities, such as the Kawada HRP4, NASA's Valkyrie and Boston Dynamics' Atlas. Examples of robot platforms designed specifically for research in cognitive and manipulation tasks are the Eindhoven University of Technology Amigo and the Willow Garage PR2, see Figure 1.3.

(a) (b) (c) (d) (e)

Figure 1.3: Several examples of advanced human-sized robots. Kawada HRP4 (a), NASA Valkyrie (b), Boston Dynamics Atlas (c), Eindhoven University Amigo (d) and Willow Garage PR2 (e).

1.1.4 Robot control program design

As described in Russell and Norvig's *Introduction to Artificial Intelligence* [153], the architecture of these robots can be characterized by two aspects in general: they sense the environment through their sensors, and act upon it through their actuators. The function that maps sensory input onto actuator output is called a robot's *control program*, see Figure 1.4.

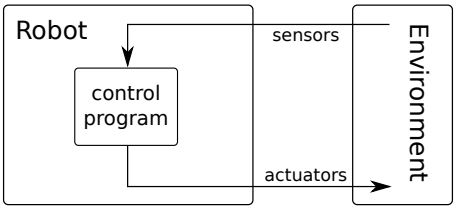


Figure 1.4: General robot interaction architecture.

How to design such a control program is a central topic of today's robotics research, and spans a broad spectrum of A.I. and software development. Related research fields include techniques for motion planning [100], locomotion [158], vision [70], tracking [193], planning [138] and knowledge engineering [132].

With modern day tools for robot control program design synthesis, these programs can be devised by manual programming up to a certain extent. This approach has similarities with planning based approaches, such as A^* [66] or *Dynamic Programming* techniques [20], by which engineers base the design of their control program on an accurate model of robot dynamics and the predicted effects of their interactions with the environment.

As robots and their desired interactions with the environment become more complex, accurate control programs are however more difficult to design manually. In robotic games, such as computer *GO*, recent design approaches involve forward-search using *upper-confidence bounds* on tree search [10] and Monte-Carlo Tree Search [35]. Such methods involve either a model of the environment, or an accurate black-box simulator that can be used for *real-time* projection, which are highly restrictive requirements to their practical application. Modeling robot control programs by manual programming can also be very time consuming and often suboptimal, because anticipation, tuning and the incorporation of all possible scenarios is tedious and often not feasible. Also, due to the static nature of such control programs, changes in the environment which are unaccounted for at design time, can cause degradation in the performance of the program.

1.1.5 Allowing robots to learn

Accordingly, as Alan Turing concludes in his 1950 paper [179] about the deployment of intelligent machines into human oriented domains, "is that the best a man can do with manual programming is injecting his own ideas into a machine through advances in engineering, but that these ideas will always fall short in comparison to the actual challenges faced." For this reason, robots should be allowed to *learn themselves* how to deal with these challenges.

In Turing's paper, a comparison is made to how a child grows up, by starting off with initially knowing nothing and learning gradually through trial and error. For robots, this is referred to as *robot learning*, and a robot architecture that supports this method is formalized in Figure 1.5.

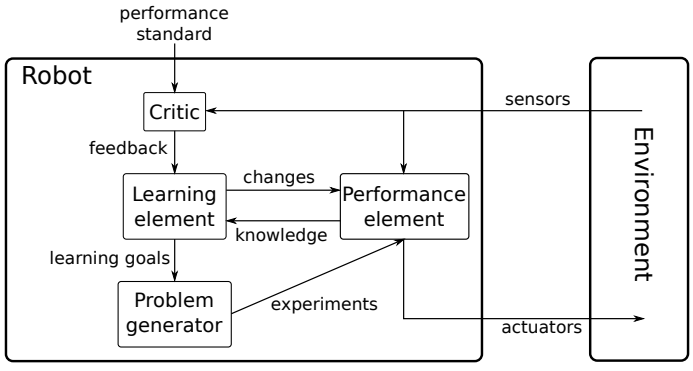


Figure 1.5: General architecture of a learning robot.

Instead of focusing on the full detailed design of a robots control program, an engineer now only has to investigate the generally much less complex design of a performance standard (often in the form of a basic cost function) that defines a robot's desired, or *rational*, behavior. Since the beginning of robotics research, robot learning has been widely applied for the development of robot control programs required for complex tasks with difficult to model interactions. Examples can be found in a wide range of applications, such as in robot locomotion [96], grasping [97] and active object categorization [148].

1.1.6 Importance of design knowledge reuse

As the injection of design knowledge into a robot's control program, either learned through experience or manually programmed, can be regarded as a highly labor intensive process [30, 77], several project groups have acknowledged the importance of robot design knowledge reuse and the design of knowledge representations that are explicitly reusable[1]. At the same time, accompanying methods are sought to store and access this knowledge through web based interfaces, making it on-line accessible for robot platforms all over the world.

A first example of knowledge standardization can be found in the GeRT [92] project, in which common representations are developed to generalize over manipulation tasks, such as serving a drink or screwing the lid of a jar. In the Rosetta project [143], robot task descriptions are designed that enable robots to operate in a combined fashion with humans. Subsequently, the Brics [2] project tries to provide generic modeling tools for robot development, which decrease the amount of required robot design time. The RoboHow [176] project enables robots to autonomously perform every day manipulating tasks by reusing task instructions found on either the web, or by observing humans. Proteus [51] and RoboDB [140] on the other hand, serve as portals for meta-data robot ontologies and robot modeling tools. Finally, in the RoboEarth [184] project, globally accessible information bases are created that allow robots to share multiple forms of knowledge that are relevant for the execution of daily tasks, such as object models, world models, environment maps and task descriptions.

Projects such as RoboEarth focus not only on the development of common representations and reusable data, but also on the storage and communication mechanisms that allow robots to share this information with each other in runtime, allowing them to execute tasks that were not explicitly planned for at design time. Examples are, for instance, the collaborative multi-robot tracking of objects and people by clustering world models, the in runtime updated performance evaluations of robot execution plans, or the sharing of object perception models and navigation maps. Databases such as those established in RoboEarth, therefore have the latest and most up-to-date information available on a variety of tasks that robots are typically required to perform in human domains.

[1] As attested by recent international robotics conference workshops on reusable robot software design, see http://www.robot.uji.es/EURON/en/software.htm

1.1.7 The idea of centralized learning and planning

Having task and environment related information available in centralized, on-line accessible storage facilities, such as RoboEarth, forwards the idea of *centralized learning and planning*; one central system, that plans tasks for thousands of connected robots, and at the same time learns from incoming data. Task allocation of robots can be based on robot capability and availability, and can be highly optimized based on time or other cost criteria. Furthermore, as this central system can be deployed within a large-scale, parallelized computing environment (such as a cloud platform [7]), robots can offload their computations to this environment, therefore requiring only lightweight client interfaces and minimal hardware interface layers to be run on board.

1.2 Objective

The objective of this thesis is therefore the investigation of methods that can be used for learning and planning, in a centralized multi-robot architecture. Additionally, supporting methods for segmentation, tracking and knowledge abstraction are investigated, as these methods enable robots to perceive their environment and to translate these percepts into machine interpretable concepts. An example of such a system is depicted in Figure 1.6.

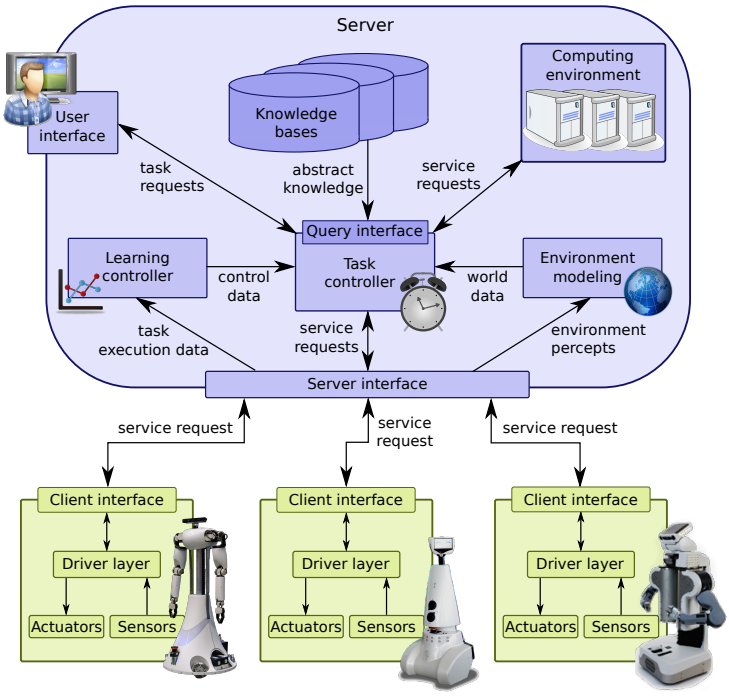


Figure 1.6: Centralized robot control architecture as discussed in this thesis.

This system should be capable of scheduling tasks for a variety of robot platforms, in an efficient and optimized manner. It should allow the interpretation of sensor data coming from robots, and be able to transform this data into machine interpretable concepts and global environment representations. Furthermore, as it is desirable that the system improves upon its own knowledge through the experience of operating robots, methods for learning will be investigated.

In this thesis, a number of required components will be investigated. A first key component that will be discussed, is an *object tracker*, that models a clear and unified overview of the environment. The goal of the object tracker is to take raw object measurements as input, and to cluster these measurements into unique object instances. Furthermore, this object tracker will be able to track objects over time, hereby dealing with object occlusions and non-linear object trajectories, such as those typically encountered in human domains. The methods that are investigated for object tracking are deployed and evaluated on a small-scale, but well controlled test platform, see Figure 1.7. This platform enables easily accessible, quantified test results and isolated performance evaluations.

Figure 1.7: Test platform used for the evaluation of tracking and learning algorithms.

On this same test platform, methods for *learning* are investigated, as they allow the system to learn from experience and to improve upon its control and decision making performance. The learning controller takes task execution data as an input, such as a sequence of actions that a robot has subsequently performed, and provides improved decision making metrics. These methods are also applied and investigated on the football table, as this platform allows us to evaluate the proposed algorithms in a well-controlled environment with fast development cycles.

A central component of the architecture discussed in this thesis, is the *task controller*, that simultaneously plans and executes tasks by interacting with server deployed algorithms and real-world operating robots. To evaluate the choice for the used planning and

execution algorithms, large scale, multi-robot tests have been conducted in unstructured household environments, with human-sized, mobile manipulation platforms, see Figure 5.9.

Figure 1.8: Two human sized mobile manipulation platforms used for testing of planning and execution algorithms deployed in human domains.

A final, crucial component is the communication framework, that allows the deployment of required components on a central server and establishes interfaces to the robots. For this, the existing cloud based computing environment *Rapyuta* has been adopted, that supports the deployment of computational algorithms in a cloud based computing environment and HTTP-based robot interfacing. An extensive read on the advantages and functionality of this framework can be found in [72].

1.3 Related work and contributions

Based on the above discussed components, this thesis will address several research areas from which applicable methods are selected, investigated, compared and if required, adapted. The concerning research areas relate to methods for object segmentation and tracking, robot learning, knowledge representation, planning and execution.

1.3.1 Object segmentation and tracking

The object tracking and segmentation methods that are discussed in this thesis are based on a well-known survey paper [193] on object tracking and segmentation design methods. This thesis presents an application of that approach on the Eindhoven University automated soccer table, where this approach is described in such a general way that it can be reused by engineers for applications in similarly unstructured and dynamical environments. The presented approach results in the selection of a suitable segmentation

algorithm, and for tracking a method was selected, that has never been applied in this type of environment before.

For tracking of *non-maneuvering* targets (i.e., targets with constant velocity), a conventional Kalman [189] filter is frequently used. For the ball that was tracked in the football table however, tracking is complicated by the ball's abrupt changes in motion, such as those typically encountered for objects moving in human domains. The targeted object can therefore be regarded as a highly agile maneuvering target: the acceleration is, for the most part, a sequence of short pulses with unknown magnitude which occur at unknown time instants, with in between nearly zero acceleration. A vast amount of literature can be found on the subject of maneuvering target tracking (MTT[2]), as attested to by the comprehensive survey [104–113] and the references therein. In the history of MTT, single model-based adaptive Kalman filtering techniques were developed first, followed by decision-based methods, which have in turn been superseded by multiple-model methods due to their superior performance and computational improvements [105, 108]. By modeling the target motion (continuous component) with different models (discrete component), the problem becomes that of hybrid state estimation.

One of the most cost-effective hybrid state estimation schemes is the sub-optimal interacting multiple model (IMM) estimator [25], which has been shown to significantly outperform a Kalman filter for target tracking in many radar applications [89, 123]. These radar systems have relatively *long sensor revisit intervals* together with relatively *slow maneuvers*. In the work of Kiru [89], research was done through simulations to see when an interacting multiple model filter is likely to have an improved estimation accuracy in comparison to a single model Kalman filter when using a constant velocity and a white noise acceleration model. The *maneuvering index*, or tracking index, is hereby used to quantify the preferred choice for these algorithms. The maneuvering index is a function of the motion uncertainty, measurement uncertainty and the sensor revisit interval, and gives an indication of when to use an IMM estimator over a Kalman filter.

Although questions have been raised regarding the validity of these results in the work of Silbert [159], the performance improvement indication still holds for the non-maneuvering time intervals (i.e., the periods after the object has changed direction), which are the main points of interest in the presented application. This thesis will show that the tracking improvements of an IMM filter over a Kalman filter in this case will show significant tracking improvements. Its low computational requirements and self-adjusting variable-bandwidth [123] make it an attractive Kalman filter alternative for this or similar applications.

1.3.2 Robot learning

Within the field of robot learning, *Reinforcement Learning* [167] especially has been closely connected to robotics research, as its abstract model of receiving rewards through environment perturbations usually directly maps onto the concept of mobile sensor/actu-

[2]Not to be confused with Multiple Target Tracking

ator platforms interacting with a priori unknown environments. Reinforcement learning requires only the experience of interacting with the environment to generate a control program, or so-called *policy*, by basically using a trial and error approach. It has been shown that policies generated by Reinforcement Learning often trump their hand-coded rivals [163]. Furthermore, recent advances in the field of Reinforcement learning have made it applicable to systems with large state-action spaces, by generalizing over states and clustering actions into condensed forms [17].

At its core, Reinforcement Learning describes how an agent autonomously generates a policy by interacting with the environment and 'remembering' what was right and what was wrong, as classified by the engineer. As this classification can be performed on a very high and abstract level, the engineer therefore does not require to have any insight on the robots internal dynamics, nor on the expected response of the environment to any of the robots induced actions. As robotic systems and their interactions with the environment are these days becoming more complex, Reinforcement Learning has become a mature and much used tool for the development of robot control programs. For this reason, Reinforcement Learning has been applied to a variety of robotics applications, such as quad-copter control [115], autonomous car driving [122] or ball dribbling [33]. In the proposed control architecture described in Figure 1.6, Reinforcement Learning can be applied as a way to improve upon task controller parameters, as performed action sequences and observed outcomes can be used as exploration data and action quality metrics respectively.

From the available existing Reinforcement Learning algorithms, this thesis targets the (stochastic) gradient based Temporal Difference method Greedy-GQ(λ) [116] because of its theoretical convergence properties, and because it scales linearly in computational complexity with the number of parameters. In this thesis, it will be compared to existing methods, such as *grid-based* Q(λ) [186], which is guaranteed to converge and has low computational costs, and to *approximate* Q(λ) [128]. The latter only has convergence guarantees under very restrictive conditions, but often converges quicker and is computationally cheaper than gradient-based algorithms (such as Greedy-GQ(λ)). According to the developers of the algorithm, Greedy-GQ(λ) has not yet been studied when applied to a large scale real-world problem, such as the one demonstrated in this thesis.

1.3.3 Knowledge engineering

To keep storage of the knowledge that is communicated throughout the centralized architecture scalable and maintainable, it is important that this knowledge is stored in efficient representations. These representations may vary from the binary data that is used for object models and navigation maps, to the abstract logic based formalisms used for planning. For the latter form of knowledge it is important to use the right level of abstraction, as this knowledge is to be used in search and planning algorithms and scalability issues arise especially for large stochastic domains, such as human environments [85]. The field of software design that concerns about this matter is the field of *knowledge engineering*.

In the field of robotics, where knowledge levels vary from low level motor control to high level reasoning, knowledge engineering often relates to the hierarchical abstraction of information. An analogy here can be found in recent studies about human psychology [131], that describe the meta-modeling of low level motion primitives into high level representations. Humans are therefore, maybe unaware, experts in *hierarchically* structuring their cognitive and manipulating abilities, hereby basing the level of hierarchical focus on the specifics of the task at hand and the environment they have to be performed in. As this hierarchical structuring allows humans to perform a wide-variety of tasks under different conditions, it has been successfully mimicked in robotics [59, 81]. It gives engineers a possibility to abstract away from time and geometrical constraints, and allows them to model, or learn, robot behaviors modularly [165], enabling the reuse of these components as part of larger problems, in possibly different tasks with different constraints.

To apply this hierarchical knowledge structuring for robots operating in human domains, it is therefore required that abstract representations of this knowledge are developed [3, 174]. As such, a certain level of *abstraction* needs to be chosen, see Figure 1.9.

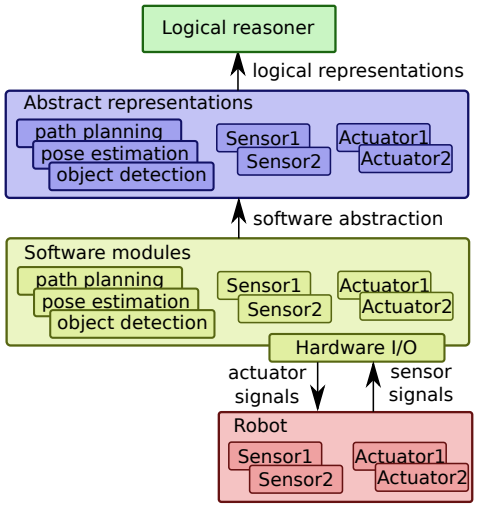


Figure 1.9: Software abstraction enabling high level reasoning.

1.3.4 Planning and execution

This abstracted task knowledge, obtained either through learning or manual programming, can be used as planner building blocks for the automated composition of robot tasks in efficient, *logic based* planning algorithms. In this light, classical planning approaches, such as STRIPS [56] like planning algorithms, have been considered for plan composition. As STRIPS planning is proven to be NP-complete and very ineffective for

large-scale domains, such as those typically found in human environments [31], work in this field started investigating more sophisticated planning methods, such as graph-based planners [26], hybrid graph-based FastForward (FF) planners [69] or Hierarchical Task Network (HTN) [55] planners. HTN planning especially, suits itself well for the use of planning in large-scale domains, by using previously composed or manually designed plans as heuristics in future planning requests. Furthermore, HTN planning allows the annotation of cost metrics to individual subtasks, enabling optimal plan selection if multiple plans exists.

This thesis therefore adopts HTN planning as a primary planning algorithm for robot tasks. Based on successes obtained in the International Planning Competition, the state of the art HTN planner SHOP2 [137] is adopted, which in this thesis derives its domain knowledge from a high level task description knowledge base. These task descriptions are called 'action recipes' in RoboEarth terminology [175], and in this work their representation is adapted, such that they allow to be interpreted by the SHOP2 planning algorithm. Although the presented work focuses primarily on the architectural design choices that are made to enable the integration of planning with subsequent plan execution, further work by co-authors in this direction can be found in [44, 119] and in Chapter 5 of this thesis.

Two other well-supported and stable architectures that are currently available for the integration of planning and execution, are the *LAAS* Architecture [21] based on the *BIP* [18] component design framework and *CLARAty* [139] developed by NASA and the Jet Propulsion Laboratory. In the work of McGann [127] the TREX control framework is adopted to control a Willow Garage PR2 service robot, allowing the robot to handle doors and plugs, while navigating using a topological map. The TREX control framework is run on top of a middle-ware called ROS [146], that provides in a vast amount of robot control and communication procedures.

Although the developers of these systems briefly mention a desire for the reuse of action *primitives*, there is no further discussion towards the reuse of hierarchical, *composite* robot plans. Also, each of these architectures has developed robot control structures that are highly dedicated to the platform at hand, impeding component reuse and task allocation through a high-level logic-based planner, such as proposed in this thesis. Also, they lack in a clear notation of how to match robot capabilities of individual platforms against required capabilities for the tasks at hand. Enabling robots with different capabilities to share composite task descriptions requires at first a common representation, that allows the matching of these shared descriptions with the platform's specific capabilities. As part of the above mentioned RoboEarth and RoboHow projects, a first implementation is described in [174], where the Semantic Robot Description Language (SRDL) [99] is used to match robot capabilities against the task related components. Chapter 5 of this thesis will exploit that concept, by using first-order reasoning to deploy required subtasks onto available robots with matching capabilities.

Section 1.3.3 discussed the upwards abstraction of software components into logical based representations, useful for the creation of abstract plans. After plan creation, descriptions need to be given on how the actions in the plan are subsequently executed on

real robots operating in real environments, a process called *action grounding*. As this thesis targets the execution of tasks on both Matlab/Simulink and ROS enabled platforms, action grounding has been applied to both middle-wares An example of a grounding ontology for ROS is given in Chapter 5 of this thesis.

1.3.5 The Semantic Web

As the design knowledge useful for the planning and execution of robot tasks in human domains can be regarded a mixture of both human- and machine generated knowledge, a preference for the representational language used in the architecture lies towards currently established machine interpretable representations, such as those used on the Semantic Web [22]. Its basic representational language OWL [141] as proposed by the World Wide Web Consortium[3] and the accompanying HTTP-based interfacing and automated reasoning tools provide in a unified framework that is capable of not just reading other sources of information, but also manipulating, updating, and combining them. These techniques allow the combined pooling of machine interpretable knowledge based on information already collected and stored by humans over the last decades, and on new information added in runtime by connected robots. A language model variant of OWL that is most suitable for use in cognitive robotic applications is OWL Description Logics (OWL-DL), as it provides in maximum expressivity but remains decidable. This allows the language to be used in most modern day reasoning modules, such as Pellet, Racer, Fact++ or in theorem proving query languages, such as Prolog and SQL. A good read on the advantages of using OWL-DL in robotics can be found in [67].

To allow web based task representations to be used as planner building blocks, extensions of OWL have been made, such as the Business Process Execution Language for Web Services BPEL4WS [6], and OWL-S [121] (formerly the DARPA Markup Language for Services, DAML-S), which extend Semantic Web representations for the explicit modeling of web services, processes and data-flows on the Semantic Web.

Based on recent efforts of the Semantic Web community to model *Everything As A Service* (XAAS) [14], a recent trend in robotics has adopted this vision for the similar modeling of robot platforms. Based on this concept, robot platforms will be modeled as on-line accessible, modularly structured collections of web services, and invoked identically to Semantic Web services as currently available. They hereby expose abstract models of their functionality, such as sensors and actuators, to service composition (or *planning*) algorithms, such as described in the work of *ubiquitous robot networks* [88], and in the integration of home automation systems and ubiquitous robots as OWL-S web services [64]. These modular robot representations can be used as plan building blocks in hierarchically structured plans, and used for plan composition by of the shelf available planning algorithms. The planning algorithms as discussed in Section 5.3.3 have therefore been applied to Semantic Web task representations for the automatic composition of web services in graph-based planning [195], hybrid graph-based FastForward (FF) planning [91] or Hierarchical Task Network (HTN) planning [55].

[3] http://www.w3c.org

As the semantics of HTN planning are based on the propositional logic Planning Domain Definition Language (PDDL) [126], it natively lacks expressivity compared to the first-order logic semantics entailed by the primarily used representation for Semantic Web services (OWL-DL) [102]. Examples are for instance the cardinality restrictions (e.g., for-all, none) that are possible in OWL-DL. Also, the complex control procedures as modeled by OWL-S, such as *while-do* and *if-then-else*, require complex mapping procedures to be represented in PDDL semantics [161] (for instance, an OWL-S while-do construct has to be mapped onto an HTN method that calls itself recursively). As it is desirable that the planning language used in the proposed control architecture has maximum expressivity, the use of a higher-order planning language, and accompanying language implementation, is preferred.

A first-order planning language that maps directly onto OWL-S Description Logics semantics, is the Situation Calculus [124, 125]. This language has been implemented in Prolog as a high-level agent programming language by Hector Levesque and Raymond Reiter, called Golog [103], and exquisitely described in Reiter's book 'Knowledge in Action' [150]. Golog allows engineers to axiomatize a Situation Calculus based action domain as primitive actions and procedures in Prolog, and the Golog interpreter will subsequently 'roll' over the procedures, hereby subsequently performing one or more primitive actions. Planning in Golog is hereby induced by the theorem solving property of Prolog, which tries to find a set of valid bindings that unify with the invoked control procedures and their parameters.

Golog however lacks in certain features that are desirable for planning in human domains. These are the possibilities for modeling sensing actions, exogenous actions, concurrency and multi-agent planning. For this reason, several extensions of Golog have been made, that enable the language to be used in high-level robot planning and execution with an increasing level of functionality. A first extension was made by Hector Levesque called *ConGolog* [45], which allowed the explicit modeling of concurrent and exogenous actions. A successor of that was IndiGolog [62], which allowed programs to be executed incrementally, based on the input obtained through sensing actions. A subsequent extension for multi-agent planning was developed by Ryan Kelly, called *MIndiGolog* [87]. The work described in this thesis builds upon this last extension, and uses its interpreter to plan for procedures that are represented in the OWL-S semantic markup language, hereby using general procedures expressed in OWL-S to be executed by multiple agents in a time-optimal fashion. As the current MIndiGolog interpreter has no interpretation mechanism for certain control procedures as defined in OWL-S, such as *any-order*, *split* and *repeat-while*, the interpreter is extended as described in Chapter 5 of this thesis.

1.4 Outline

This thesis will start of with a detailed description of one of the primary robotic platforms used for research and demonstration in this work, and as a public demonstration platform for Eindhoven University in general; the Eindhoven University automated football table. In this first part, hardware design choices for this platform will be discussed, together

with an extensive description of a model-based method used to detect and track a primary object of interest.

The chapter hereafter will continue with the football table as a robotic research platform, and here a state of the art method for robot learning will be discussed and evaluated.

The validation domain in Chapter 4 will shift from the football table to the service robot domain, where now methods for hierarchical plan composition and execution will be implemented on the Eindhoven University Amigo robot and evaluated in a real household environment.

Subsequently, Chapter 5 will discuss an extension into this domain, where now existing representations from the Semantic Web are used for plan representation, and a more expressive language will be used for plan execution. This chapter will also describe a more clear separation between representation, plan composition, execution and grounding, by defining an ontological based component model for the low level grounding layer. The conducted tests in this chapter, will entail a two-robot experiment, where these robots will be fully controlled by a cloud deployed planning and execution platform.

This thesis ends with a concluding section, in which both the achievements and limits of the work as well as the opportunities for future research will be discussed.

Chapter 2

Object Localization And Tracking

This chapter presents the development of an object localization and tracking algorithm, that is to be applied in environments with high dynamics and fast update rates. The described approach is based on an earlier survey paper on object segmentation and tracking, where a general selection procedure for applicable techniques was proposed. This chapter describes why these techniques are not well suited for our specific application. As a solution, an IMM estimation technique is adopted that has not been applied in this type of context before. To evaluate the IMM estimator in a well-controlled experiment, it is applied to the Eindhoven University automated football table and compared to a commonly used and carefully tuned Kalman filter.

This chapter is based on *"Ball Localization And Tracking In A Highly Dynamic Table Soccer Environment"*, R. Janssen, M. Verrijt, J. de Best and R. van de Molengraft, Mechatronics Special Issue on Visual Servoing, 2012

2.1 Introduction

The last decade a lot of research has been performed on the deployment of cooperative multi-agent systems in unpredictable and unstructured real-world environments. Examples can be found in robots that collaborate with humans in maintaining large warehouses [63], human-robot assembly lines [172], underwater robots that operate as single sensor networks [183] and swarms of unmanned ground vehicles that perform reconnaissance missions in large disaster areas [82].

A key element in these multi-agent systems is the applied control strategy, and evaluation of such a strategy in an everyday unstructured environment is an extremely daunting task. Large scale experiments need to be conducted that require time, logistic and financial effort, multiple sensors have to be repetitively calibrated and most importantly, the environment is likely to respond in an uncontrollably hazardous manner, hereby endangering the costly agents. Therefore, simplifications are sought to manage complexity by focusing on handling the dynamics, where the environment is chosen to be controllable and well structured.

This kind of abstract simplification can for instance be found in the RoboCup Mid-size League [49], where two teams of robots compete against each other in a game of field soccer. This concept transforms the hazardous large scale outdoor test environment into a controllable indoor game-setting, allowing researchers to easily tune their strategies, redesign their components on the spot and assuring that each of their costly agents will survive the conducted tests. Although the RoboCup setting is significantly less complex and less hazardous than the outside world, there are still factors involved that limit repeatability and efficiency of testing:

- each team consists of at least five autonomous robots. These complex mechatronic devices typically need a lot of maintenance, and an extensive infrastructure for hard- and software management,
- varying inter-robot communication delays make it difficult to execute commands in sync, which may result in non-causal and undesired behaviors,
- calibration discrepancies between agents in their perception modules may result in different beliefs about the state of the environment.

To address the above issues while maintaining the highly dynamic character of the soccer environment, a professional soccer table has been acquired that on one side has been equipped with electro-mechanically controllable rods, see Figure 2.1.

Figure 2.1: The Eindhoven University automated soccer table.

This system allows the execution of a multi-agent strategy within the highly dynamic characteristics of the table, represented by the fast moving ball and the human- and machine-controlled puppets. An extended overview on the design concepts of the table can be found in [78][1].

With this set-up, the above mentioned problems encountered in the RoboCup Mid-size league context are addressed as follows:

- only one system needs to be maintained,
- no wireless communications are necessary and all signals go through one computer, so inter-agent communication delay will be negligible,
- only one system needs to be calibrated,
- there is only one perception module, and thus only one belief about the state of the environment.

The concerning object in our setup is the ball, and the environment furthermore consists out of two main components,

- The fully controllable, mechanically actuated puppets,
- Their uncontrollable and (at this stage) unobservable adversaries (which are controlled by the humans),

The states of the actuated puppets consist of their position and orientation. These states can be easily derived from the incremental encoders used in the motion control loops. These control loops are run on a dedicated PC, and steer the puppets by actively rotating and translating the actuated rods, see Figure 2.2.

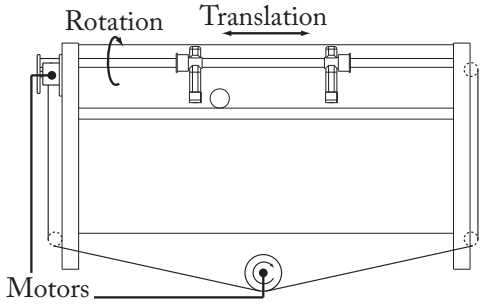


Figure 2.2: Position control of the actuated puppets.

[1] An explanatory video about the table can be found at `http://youtu.be/0qE_a0wFRa0`

The states of the ball consist of the ball's position and velocity, which are required to determine the ball's heading. These states however, can not be directly measured. They have to be reconstructed from noisy position measurements, which will be obtained by a camera above the field, see Figure 2.3.

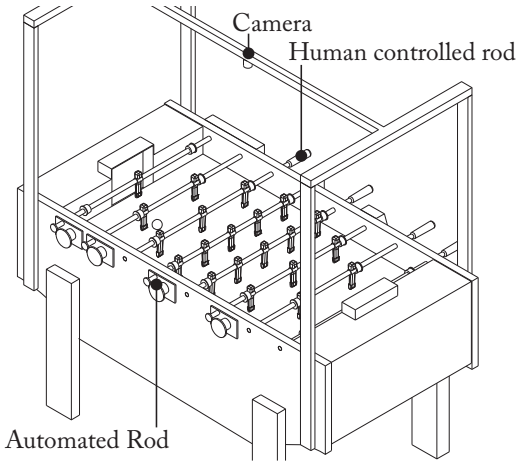


Figure 2.3: Schematic representation of the football table.

Localizing the ball in the captured images is not trivial though, see Figure 2.4, as there are other objects present that,

- Fully or partially occlude the ball,
- Resemble the ball in either shape, color, or both,
- Visually merge with the ball, making it difficult to separate one from another.

In this chapter, therefore a survey paper on object localization and tracking [193] will be consulted, from which a set of well-known computer vision localization techniques will be derived to form a robust detection classifier. From these detections the full state of the ball will be reconstructed by using a *state estimator*. For the choice of the state estimator a commonly used method is compared to a method that we have carefully selected for this special application. We would like to state that this selected method, to our best knowledge, has not been applied in this type of highly dynamic context before.

The outline of this chapter is as follows. In the following section related work on similar applications will be presented. Hereafter, some general research within the field of object detection and state estimation techniques will be discussed, from which the choices for our preferred methods will follow. Two state estimation techniques will be compared in

a simulation experiment, which will allow us to adequately compare their performance. The preferred method resulting from these tests will also be implemented and evaluated on the real table soccer set-up. We will conclude with a discussion and a conclusion on the solution to our problems. Extended information that is regarded as trivial to the general understanding of this chapter can be found in Appendix A.

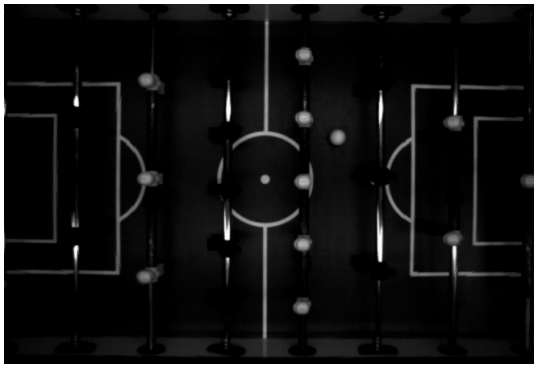
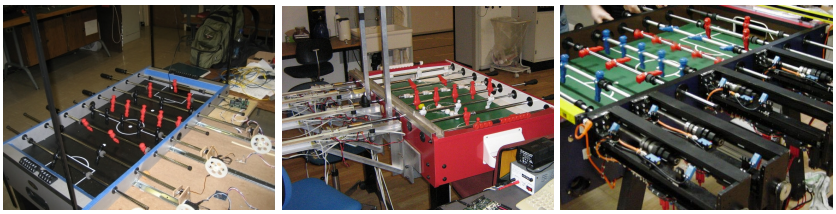


Figure 2.4: Typical image captured by the overhead camera.

2.2 Related Work On Similar Applications

For entertainment and computer vision research purposes, several other project groups have already faced the challenge of developing a robotic table soccer adversary, see Figure 2.5.



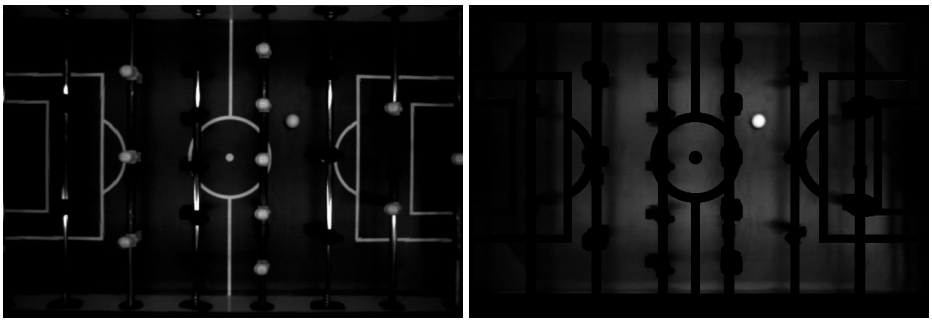
(a) (b) (c)

Figure 2.5: Georgia Tech (a), DTU (b) and KiRo (c) projects.

While the methods to actuate the puppets seem to be more or less alike, the methods to detect the ball mostly differ. At the University of Adelaide [36], a robotic soccer table has been developed in which a laser grid is mounted underneath the feet of the puppets. In this way, the ball is the only object that can breach the grid. When it bounces over

the field however, and thus over the grid, it is momentarily undetectable. This problem also exists in the DTU [136] and StarKick [188] projects, where the field is replaced by a semi-transparent plate. By mounting a camera underneath the plate a shadowed projection of the ball can be detected, but this detection fails when the ball bounces upwards. In both the KiRo [187] and Georgia Tech [41] projects a straightforward color segmentation method is used to segment the ball from the environment. Currently this seems to be the most promising approach, although the acquisition time for both capturing and processing of color images can be assumed to be larger than for mono images, hence degrading the ball tracking performance and increasing the computational load. Due to the limited documentation on these projects, no information can be found on their applied state estimation techniques.

In earlier work on our set-up [79], a method was presented that describes how a dynamically calculated mask can be used to detect the ball in the captured images. By combining a *static mask* for the field markings with a *dynamic* mask for the moving puppets, all the objects that resemble the ball can be filtered out, leaving only the ball itself visible in the images. In Figure 2.6 a picture is presented showing the outcome of this mask, next to the original image captured by the overhead camera. For state estimation, in this set-up a straightforward, but carefully tuned Kalman filter was used



(a) (b)

Figure 2.6: Captured image (a) and applied masking methods (b).

Although this combined method of localization and tracking has proven to work well, careful investigation has shown us that on both aspects improvements can be made. Regarding the dynamic mask that is used for localization, we have to conclude that it is highly sensitive to calibration errors. The main reason is that when the masks are not exactly aligned during the initialization phase, at runtime the localization algorithm will mistakenly assume (part of) the puppets as ball positives. The fact that, due to wear and tear of the actuated control rods, calibration and initialization have to be performed repetitively make it a tedious enterprise.

Regarding the Kalman filter that was used for state estimation, we found that although the bouncing ball does not entail a linear system (one of the major assumptions for this type of state estimator), it performed reasonably well. At that point however, the Kalman gains were tuned based on a minimization of the position error. When also requiring

convergence on the velocity (to obtain an accurate heading estimate) , convergence times increased drastically, resulting in slower response times and an increase in the number of missed intercepts.

Therefore in this chapter both detection for localization, and state estimation for tracking will be re-investigated, to see if improvements can be made by exploiting different techniques. Although the presented methods for localization are commonly used in the field of high speed computer vision, a different area of application is consulted for state estimation. The concepts of this technique and why, to our opinion, it can be used for our application will be explained in the following section.

2.3 Method Selection

In this section, we construct an object tracking technique that is based on a well-known survey paper on object tracking [193]. The bottom-up approach presented in this paper allows for a better understanding of the encountered challenges and a guided method to tackle them. According to this approach the topics that need to be addressed are (from bottom to top):

– Object Representation (Bottom)
  ⌙ Feature Selection
    ⌙ Object Detection
      ⌙ Object Tracking (Top)

The remainder of this section will discuss each of these topics. The reader is referred to [193] and references therein for examples, and in-depth evaluations and surveys of the following techniques presented.

2.3.1 Object Representation

Depending on the application, an object can be represented by its shape and/or its appearance, see Table 2.3.1.

Representation	Common usage
Point(s)	Small objects
Primitive geometric shapes	Simple rigid objects
Object silhouette and contour	Complex nonrigid shapes
Articulated shape models	Articulated objects
Skeletal models	Object recognition

Table 2.1: Object shape representations.

Note that these situations do not represent a restriction. Skeletal models for instance, can also be used for rigid or articulated objects. An *object's appearance* can also be represented using its features, for example, color and/or texture. The appearance representations mainly differ in the way these features are used and are often combined with a shape representation. Table 2.2 shows a number of common appearance representations together with some of their characteristics.

Representation	Characteristics
Probability densities	· Computed from a specified interior region, e.g., a shape. - Parametric: Gaussian and Gaussian mixtures. - Non-parametric: Parzen windows and histograms.
Templates	· Object appearance is generated from a single view. · Contains both spatial and appearance information.
Active appearance models	· Object shape and appearance are learned from training. · Uses landmarks with an appearance information vector.
Multi-view appearance models	· Encodes multiple views of an object. · Appearances in all views needed ahead of time.

Table 2.2: Appearance representations

In a game of table football, the primary object of interest is the ball. A suitable *shape representation* is of course a circle (primitive geometric shape), which essentially decomposes into a point and a radius. A more complex shape representation hardly gives any additional information. As for the *appearance representations*, the active and multi-view models are believed to be unnecessarily complex. The probability density and template representations seem to be reasonable. However, they are commonly used in combination with some sort of matching algorithm, which increases the computational complexity.

The most popular method for circle detection in images is the Circle Hough Transform (CHT) [52], that extends the basic idea of the Hough transform for line detection to that of circles. The original CHT is known to be robust to noise and occlusions, with the main drawback being its computational complexity [171]. A number of techniques have been developed trying to overcome the computational complexity issues, while maintaining the robustness of the original CHT method. Due to the large number of variations, it is almost impossible to refer to all the works available on this subject. The reader is referred to the comprehensive overviews and comparisons found in [75, 86, 90, 101, 194] for more information.

The speed of the discussed circle detection algorithms are, among others, dependent on the number of edge pixels found and the size of the parametric space. As for the football table, there is only one ball that is relatively small, which results in only a few edge pixels. Furthermore, the size of the ball doesn't change so the parametric space can be well defined beforehand. This makes it easier to implement the above algorithms and also limits the computational effort required to run them. Because of this, it may be possible for the original CHT to run real-time in this situation. If not, one of the alternatives presented may prove to be applicable.

2.3.2 Feature Selection

The most commonly used visual features are described here. Most tracking applications employ a combination of features for better performance.

Color is one of the most widely used features, mainly due to its easy implementation and the variety of color spaces available, for example:

- RGB Red, Green, Blue
- HSI Hue, Saturation, Intensity
- YUV Luminance (Y), Chroma (U,V)

A color space with a separate intensity channel like HSI or YUV is typically more robust with respect to variations in *intensity*. A color can hereby be described by the remaining two values, spanning all intensities. There is no definitive answer on which color space to use; it is entirely an image/application dependent question [181], i.e., dependent on several factors, including the used hardware and the object(s) to be detected.

Edges are less sensitive to illumination changes than color, due to objects generating a strong change in intensity at their boundaries even if illumination varies. Note that the sensitivity to a change in illumination depends on how the edges are obtained; a gradient method is far less sensitive than obtaining an edge via colored blobs for example. Edges are commonly used as features in applications that track the boundary of objects, e.g., when a contour representation is used.

Optical flow is used as a feature when something is interesting based on its motion, or when an object's motion is exactly what makes it interesting. It is characterized by a field of displacement vectors representing the distance a point has moved between two frames. These algorithms generally rely on the assumption that an object doesn't change its appearance as it moves. There are two approaches to optical flow, i.e., *dense* and *sparse* methods. In dense optical flow the field is calculated for each pixel in some region. High computational costs are involved in these algorithms to solve for the ambiguous pixels. In sparse optical flow a subset of points, that are somehow specified beforehand, are tracked. This involves selecting points with "good features to track". Corners, for example, have an expressive local texture which can be used. The sparse method thus employs a combination of features.

Using ***Texture*** as a feature requires descriptors to be generated. This can be done manually, or automatically, for which a variety of algorithms exist. As with edges, these features are not as sensitive to illumination changes as color.

Although automatic feature selection algorithms exist, the features are chosen manually in most situations. A manual selection is also believed to be sufficient in the case of the football table, because of the well specified knowledge of the scene. The interested reader is referred to [193] for an introduction to automatic feature selection. Each of the features presented in this section seem to be appropriate for use in the football table setup. However, dense optical flow has high computational costs, and a uniform white ball does

not really have any "good features to track", for sparse optical flow. An attempt was made for a similar situation (orange ball) in [144], where it was reported that such a method gave very poor results. Concluding on the above summary, a combination of color, edges, and/or texture seems to be the most promising approach for a fast localization of the ball.

2.3.3 Object Detection

Object detection in every frame or when the object first appears in a scene is required in every tracking method [193]. Most object detection methods use information acquired from single frames only. Methods that use temporal information acquired from a sequence of frames to reduce false detections, are also available. A common approach to this is some form of frame differencing. The main categories of object detection for the purpose of object tracking are explained below. Popular representative methods within these categories are shown in Table 2.3.

Category	Representative Methods
Point Detectors	· Moravec's Detector · Harris Detector · Scale Invariant Feature Transform · Affine Invariant Point Detector
Segmentation	· Mean-Shift Clustering · Graph-Cuts · Active Contours
Background Subtraction	· Mixture of Gaussians · Eigenbackground · Wallflower · Dynamic Texture Background
Supervised Learning	· Support Vector Machine · Neural Networks · Adaptive Boosting

Table 2.3: Detection methods

Point detectors are used to find so called interest points. These points have a local texture that is characterized in some predefined way, e.g. via intensity variation. A desirable quality of such a point is its invariance to changes in illumination and camera viewpoint [193].

Segmentation algorithms aim at dividing an image into regions, or partitions, that are perceptually similar. To this end, two main issues have to be addressed: A criterion for a good partition and a method for achieving efficient partitioning [193].

Background subtraction is the process of finding significant regional differences between each incoming frame and a background model. These regions signify a moving object and are marked for further processing.

Supervised learning mechanisms can be used to perform object detection by learning different object views automatically from a set of examples. The learning examples are composed of manually selected pairs of object features and an associated object class. A hyper-surface is computed that separates the different object classes in a high dimensional space. Such methods usually require a large collection of manually labeled samples from each object class.

A uniformly colored rolling ball does not exhibit points that have an expressive local texture. If the background is uniformly distinct, an exception would be the edge-points. Even then, they are much easier detected with an edge detection algorithm. Segmentation of the image into perceptually similar regions seems as an easy and efficient approach for the football table. Due to the well specified knowledge about the scene, the two main ingredients for good segmentation are present. This can be done with a good combination of features, in particular color. A more detailed discussion on color image segmentation is given in [171]. Background subtraction is a nice approach for situations with static cameras, such as this. These methods seem to be applied mostly in situations with a relatively low frame-rate, a common example being traffic cameras. Due to the moving puppets, a dynamic background model that requires to be updated is necessary, as was proposed in [79]. Furthermore, a high frame rate is desired, because of the highly dynamic nature of the game. The increased complexity and computational cost of such methods is believed to be impractical and unnecessary for this particular application. Supervised learning is also believed to be a rather unnecessary complex approach to the problem at hand.

2.3.4 Object Tracking

The goal of an object tracker is to reconstruct the object's trajectory over time by locating its position, and possibly obtaining its region information, in each frame. The object detection method as discussed previously, as well as the correspondence between object instances across frames (data association) need to be performed to achieve this. This can be done separately or jointly. In the first case possible objects are obtained by a suitable detection algorithm. The tracker's job is to then associate the possible objects to their respective tracks. In the second case, the object region and data association is jointly estimated by iteratively updating object location and region information obtained from previous frames [193]. In addition to the data association problem, state estimation can be performed. This can also be done separately or jointly. When the state estimation is done separately, some sort of deterministic data association has taken place. When tracking a single object in noise or in the case of multiple objects, a joint solution of data association and state estimation is often required. For tracking in the context of vision applications, three main categories can be distinguished: *point tracking*, *kernel tracking* and *silhouette tracking* [193].

These categories can be split up into subcategories, revealing the variety of methods available. Common tracking methods used in vision, also from [193], are described together with relevant applications in Table 2.4.

Table 2.4: Tracking methods

Sub Categories	Representative Methods
	Point Tracking
Deterministic	· Modified Greedy Exchange · Greedy Optimal Assignment
Probabilistic (Statistical)	· Kalman Filter · (Joint) Probabilistic Data Association Filter · Probabilistic Multiple Hypothesis Tracking
	Kernel Tracking
Template & Density Based	· Mean-shift · Kanade-Lucas-Tomasi · Layering
Multi-view Based	· EigenTracking · Support Vector Machine
	Silhouette Tracking
Contour Evolution	· State Space Models · Variational · Heuristic
Shape Matching	· Hausdorff · Hough Transform · Histogram

Tracking the center of the ball gives enough information for a successful intercept. Its spacial characteristics don't give any other relevant information, as the ball's radius doesn't physically change. Point tracking is then obviously appropriate for this particular application.

Maneuvering Targets

For non-maneuvering targets a conventional Kalman filter is frequently used for tracking. In a game of table football however, the tracking is complicated by the ball's abrupt changes in motion. The ball can therefore be regarded as a highly agile maneuvering target: the acceleration is, for the most part, a sequence of short pulses with unknown magnitude which occur at unknown time instants (maneuver, M), with in between nearly zero acceleration (non-maneuver, NM). A vast amount of literature can be found on

the subject of maneuvering target tracking (MTT[2]), as attested to by the comprehensive survey [104–113] and the references therein. In the history of MTT, single model-based adaptive Kalman filtering techniques were developed first, followed by decision-based methods, which have in turn been superseded by multiple-model methods due to their superior performance and computational improvements [105, 108]. By modeling the target motion (continuous component) with different models (discrete component, e.g., M/NM), the problem becomes that of hybrid state estimation. A major challenge arises from the target motion-mode uncertainty, i.e., the uncertainty in determining which mode is in effect at a particular time instant. If the target motion, or base state (x_k), behaves linear with respect to the system mode (s_k), it can be represented by a Markov jump-linear system (MJLS) [108]:

$$x_{k+1} = F_k(s_{k+1})\,x_k + w_k(s_{k+1}) \tag{2.1}$$

$$y_k = H_k(s_k)\,x_k + v_k(s_k) \tag{2.2}$$

Under the Gaussian noise assumption, the optimal estimator for this system is a Gaussian mixture with an exponentially increasing number of terms [16], making it impractical to implement. One of the most cost-effective hybrid state estimation schemes is the sub-optimal interacting multiple model (IMM) estimator [25], which has been shown to significantly outperform a Kalman filter for target tracking in many radar applications [89, 123]. These radar systems have relatively *long sensor revisit intervals* together with relatively *slow maneuvers*. To the authors' knowledge, the use of such an estimator in high speed computer vision for tracking highly agile maneuvering targets has not yet been investigated. Its low computational requirements and self-adjusting variable-bandwidth [123] make it an attractive alternative for such applications.

In [89], research (through simulations) was done to see when an interacting multiple model filter is likely to significantly improve estimation in comparison to a single model Kalman filter when using a constant velocity (CV) and a white noise acceleration (WNA) model. The *maneuvering index*, or tracking index, is used to quantify this choice. This coefficient is a function of the motion uncertainty (σ_w), measurement uncertainty (σ_v) and the sensor revisit interval (T). It gives an indication of when to use an IMM estimator over a Kalman filter. For a *piecewise constant* WNA model the maneuvering index is given by

$$\lambda = \frac{\sigma_w T^2}{\sigma_v} \tag{2.3}$$

In [89] it was shown that

> *"...when the underlying true target motion model has WNA, above a maneuvering index of 0.5 an IMM estimator is preferred over a Kalman filter to track the target motion."* [89]

Although questions have been raised regarding the validity of some of the found results [159], the performance improvement indication still holds for the non-maneuvering time

[2]Not to be confused with Multiple Target Tracking

intervals (i.e., the periods after the ball has bounced), which are the main points of interest in this application. Denoting the maximum acceleration in a single time-step as $\bar{a}$ and assuming that (next to the Gaussian assumption) there is a 99% certainty (3σ) the acceleration is in the range $[-\bar{a}, \bar{a}]$, the process noise standard deviation is given by

$$\sigma_w = \frac{\bar{a}}{3} \tag{2.4}$$

Similarly, assuming a 99% certainty that all measurements originate from some point on the ball, the measurement noise standard deviation is given by

$$\sigma_v = \frac{r_b}{3} \tag{2.5}$$

Substituting equations (2.4-2.5) in (2.3.4), and using $\lambda = 1/2$ the lower limit of the maximum acceleration can found for which the IMM filter will likely improve performance.

$$\bar{a}_{\text{IMM}} = \frac{r_b \cdot f_{vision}^2}{2} \tag{2.6}$$

Using $r_b = 17.5$ [mm] and $f_{vision} = 200$ [Hz] a value of $\bar{a}_{\text{IMM}} = 350$ [m/s^2] is found. This corresponds to a velocity change (of the ball) between two consecutive time-steps of 1.75 [m/s]. Noting, for example, that at a speed of 2.5 [m/s] the ball can (nearly) instantaneously change to a speed of -2.5 [m/s], an IMM estimator could improve performance. At lower sampling frequencies this improvement will become even more profound.

2.4 Implementations

In the previous section, a plethora of methods have been discussed for both localizing and tracking. This section takes the outcome of that discussion as a starting point for an implementation on the real table soccer set-up. From the previous section follows that the proposed tracker can be described as in Table 2.5

Table 2.5: Proposed tracker

Issue	Choice
Representation	· Circle (Primitive geometric shape)
Features	· Color · Edges
Detection	· Segmentation
Tracking	· Point tracking

2.4.1 Localization

In Section 2.3.3, a discussion on methods and issues involved in target localization was presented. A few words were also devoted to their application to the football table. Here, a relatively simple, but efficient, localization method is described. It is a mix of components taken from the existing method and the "real-time color ball tracking" method, discussed in [171], together with some additional components. Instead of using monochrome images as was proposed before in [79], now YUV color images are obtained from the camera. This gives additional information which is used to find the ball in a more robust fashion. Note that the Y-channel of an YUV image is in effect a monochrome image, making chroma (U,V) the additional information.

For obvious reasons, the static mask and region of interest are retained from the existing algorithm. The rest of the algorithm works as follows. A simple 3D color classifier for the ball is manually obtained (off-line) using several exemplar images. This classifier is used at runtime to check which (non-masked) pixels belong to the ball's color class, resulting in a binary image. In Figure 2.7 a snapshot is given of our color calibration tool, indicating the different color classes.

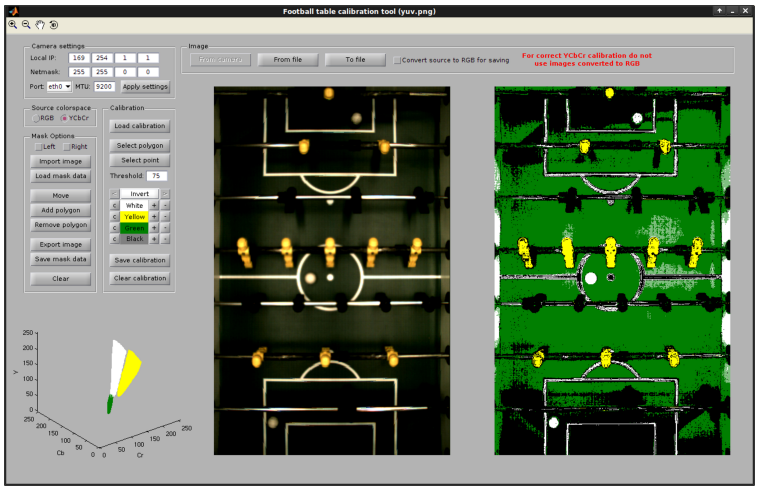


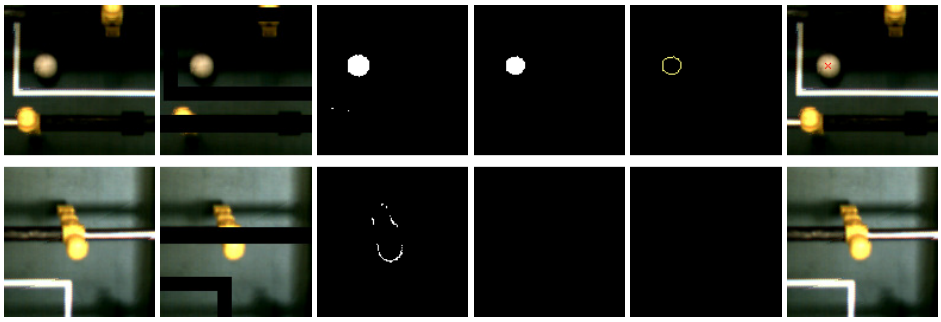
Figure 2.7: Snapshot of our calibration tool indicating the different color classes.

After color segmentation, an erosion operation is performed to remove noise pixels, and from the resulting image the *outer* contours (boundary pixels) are found.[3] Each of the found contours are subsequently checked to see if they meet the following requirements:

- Its bounding box size (width/height) must be in a specified range.
- Its length (number of pixels) must be in a specified range.

[3]This is done with the Open-CV algorithm `cv::findContours`, based on [170].

The resulting set of contours is then checked once more, where the bounding box must be smaller than a predefined square, i.e., the smallest square that can encapsulate the entire ball anywhere in the image. If this requirement is met, the CHT algorithm is applied to find the ball center. Recall from the previous section that the CHT algorithm is robust with respect to noise on the circle radius and occlusions. The sequential steps for two different input images are depicted in Figure 2.8.



(a) (b) (c) (d) (e) (f)

Figure 2.8: Steps performed for ball localization. Input image (a), static masking (b), classification (c), erosion (d), contours (e) and CHT center-point (f).

Due to lighting, and the camera's sensor, non-ball and ball pixels can have a similar color. In particular, the edges of the (white) ball and (yellow) puppets are hard to distinguish in some situations. These ambiguous regions are preferably left out when obtaining the classifier. A more conservative calibration results in less noise, but also less of the ball being detected (see Figure 2.9). Even then, situations still occur where non-ball pixels are marked as ball pixels. The noise this creates is dealt with by the elementary operations, the contour checking and the CHT algorithm. Note that the puppet and thus the ambiguous pixels can be removed by using the dynamic mask from the existing method described in [79]. Although this isn't a necessity, it does allow for a less conservative color classifier to be used.

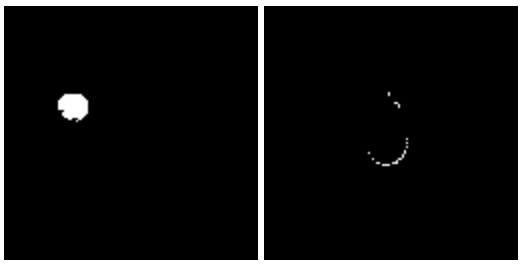


Figure 2.9: A more "conservative" classifier used on Figure 2.8(b).

The colors that show up in the image are influenced by the ambient light and the reflective properties of surfaces. In general, a different color classifier is needed whenever the ambient light significantly changes. The camera gain setting and the lights (with dimmers) just above the camera can also be used to try to adapt to new situations. The classifier essentially determines how well (and if) the localization algorithm works. Using color makes the ball localization a lot more resistant to wrong detections. Only (close to) white objects will now be detected instead of all high intensity objects. Furthermore, the objects are limited by size and perimeter.

This method allows for a more robust localization of the ball, in comparison to the existing method. The chance of incongruous objects creating wrong detections is somewhat minimized. Note that if a close to white object with the right dimensions and properties enters the scene a wrong detection can still take place. Due to the unlikeliness of such a situation, and the need for the tracker to be reset often this is acceptable. The need to do any data association is thus effectively removed. Also, the adaptivity of the proposed IMM estimator can allow for it to "correct" itself when a wrong measurement enters the filter. This localization method has proven to work very well under different lighting circumstances (when configured correctly) at symposiums and open days.

2.4.2 Tracking

In Section 2.3.4 methods and issues involved with target tracking were discussed. It also included some reasoning as to what is likely to improve tracking in the football table application. For this application, the IMM algorithm discussed in this section should allow for a significant improvement in comparison to a single model Kalman filter, while maintaining a low computational cost. A performance comparison of the two's use is proposed in [89, 159] and validated for the football table by simulations in Section 2.5. The values for the design parameters (covariances and transitions probabilities) are obtained in these sections through experimentation, although they can also be obtained by using Genetic Algorithm (GA) optimization techniques [24]. In this section, the model-set determination and some implementation issues are discussed. Another possibility is the use of a nonlinear model in combination with a particle filter, or UKF. Such a method requires accurate modeling of collisions. Here, the choice was made to use an IMM estimator based on: the ball's behavior belonging to a MJLS, the improvement indications and low computational cost.

Model-set
Two models are used to handle both situations of non-maneuver ($j = 1 = \text{NM}$) and maneuver ($j = 2 = \text{M}$). The non-maneuver model is naturally a constant velocity model. Due to the stochastic nature of the acceleration, a piece-wise constant white-noise acceleration model is used for maneuvering instances. The j-th mode obeys

$$x_{k+1} = Fx_k + \Gamma \tilde{w}_k^{(j)} \tag{2.7}$$

$$y_k = Hx_k + v_k \tag{2.8}$$

where

$$F = \begin{bmatrix} F_x & 0 \\ 0 & F_y \end{bmatrix} \qquad F_{x/y} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \tag{2.9}$$

$$\Gamma = \begin{bmatrix} \Gamma_x & 0 \\ 0 & \Gamma_y \end{bmatrix} \qquad \Gamma_{x/y} = \begin{bmatrix} T^2/2 \\ T \end{bmatrix} \tag{2.10}$$

$$H = \begin{bmatrix} H_x & 0 \\ 0 & H_y \end{bmatrix} \qquad H_{x/y} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.11}$$

where $\tilde{w}_k^{(j)} = \begin{bmatrix} \tilde{w}_{k,x}^{(j)} & \tilde{w}_{k,y}^{(j)} \end{bmatrix}^\mathsf{T}$, $v_k = \begin{bmatrix} v_{k,x} & v_{k,y} \end{bmatrix}^\mathsf{T}$ are zero-mean white noise processes with covariance matrices

$$\tilde{Q} = \mathrm{cov}(\tilde{w}_k^{(j)}) = \begin{bmatrix} \sigma^2_{\tilde{w}_x^{(j)}} & 0 \\ 0 & \sigma^2_{\tilde{w}_y^{(j)}} \end{bmatrix} \qquad R = \mathrm{cov}(v_k) = \begin{bmatrix} \sigma^2_{v_x} & 0 \\ 0 & \sigma^2_{v_y} \end{bmatrix} \tag{2.12}$$

$$Q = \Gamma \tilde{Q} \Gamma^\mathsf{T}$$

in which $\sigma_{(\cdot)}$ denotes the standard deviation of $(\cdot)$. When a single model is used (for the Kalman filter) the (j) superscript can be removed, i.e. a single WNA model is used for both maneuver and non-maneuver instances.

Issues

Two important issues when tracking the ball on the football table are shortly discussed here: *Complete occlusion* and *track initiation/termination*.

Occlusions in vision applications are commonly dealt with by using a dynamic model to keep predicting the object's location until the object reappears [193]. This is simply propagating the last known *a posteriori* estimate through the "time update" equations until the ball reappears. The ball may bounce during such an occlusion, so the wise choice for propagation of the covariance is the white-noise acceleration model. If the ball comes to a standstill while it is occluded and remains there for some time, the ball will effectively be lost when using such a solution.

Track initiation and termination is needed because the ball can exit and re-enter the scene at different locations. This can be due to a goal being scored, or the ball being lost due to occlusions. Due to the ball being the only target tracked, the initiation/termination can be done by simply resetting the filter. Deciding when this reset has to be performed must also be implemented. A simple solution is applied using a time-out starting when the ball is lost. Once the time-out has elapsed without finding the ball, the filter isn't used anymore. At the time the ball is found again, the filter is reset with its initial position at the ball location, zero velocity and a pre-set covariance.

2.5 Simulations

Before testing on the real set-up, a quantitative evaluation is performed to determine if the IMM estimator will indeed improve performance compared to the Kalman filter currently applied to our application. The emulated ball trajectory used for this simulation contains speed changes that are obtainable in a real game of table football, see Figure 2.10. Because we assume motion in the $x-$ and $y-$ directions to be independent of each other, it suffices to focus on one direction only (x in this experiment).

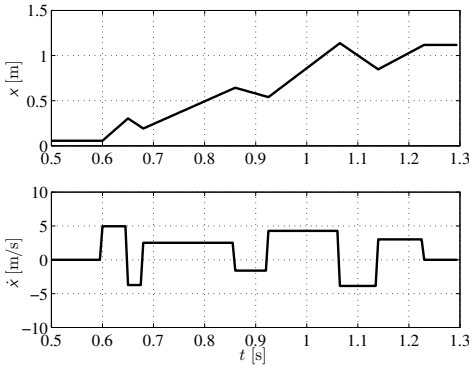
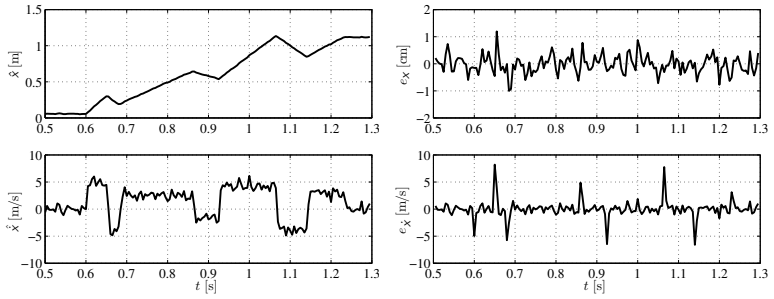


Figure 2.10: Simulated ball trajectory

The simulated measurement noise is set at the experimentally found value of $\sigma_v = 2$ [pix], which corresponds to 3.5 [mm]. The sampling frequency used is 200 [Hz].

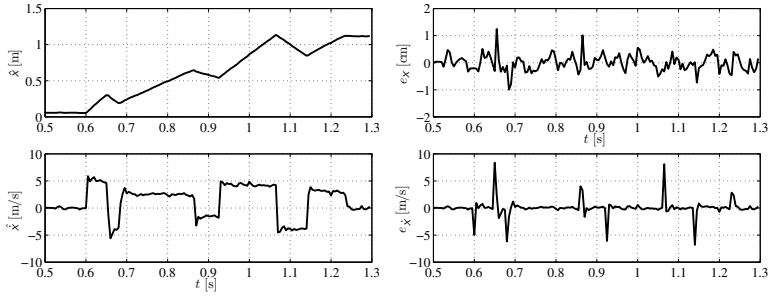
The Kalman filter currently used, with a single white noise acceleration model, is compared to an IMM estimator with two models: A constant velocity (CV) model for non-maneuver, and a white noise acceleration (WNA) model for maneuver. We note that the proposed WNA model, see Section 2.4.2, is the simplest maneuvering model and that other more involved models are also possible [106]. Both filters are designed by selecting parameters that minimize the root mean square errors (RMSE) of the velocity $e_{\dot{x}}$, hereby assuring an accurate heading estimation required for intercepting the bouncing ball.

In Figs. 2.11-2.12 and Table 2.6 the time- respectively RMS- errors for both position and velocity are shown, which indicate that the IMM has improved performance for the non-maneuvering motion. In comparison to the Kalman filter, the IMM estimator produces smoother velocity estimates for non-maneuvering, while maintaining the ability to react to the maneuvers quickly. This is due to the IMM's self-adjusting variable-bandwidth, see Section 2.3.4.



(a) Estimates (b) Errors

Figure 2.11: Kalman filter simulation results



(a) Estimates (b) Errors

Figure 2.12: IMM estimator simulation results

Table 2.6: Estimation errors

Estimator	Overall RMSE		NM† RMSE		Peak error	
	x [mm]	$\dot{x}$ [m/s]	x [mm]	$\dot{x}$ [m/s]	x [mm]	$\dot{x}$ [m/s]
Kalman	3.4	1.6	3.2	0.63	12	8.2
IMM	2.9	1.5	2.5	0.21	12	8.4

† Non-maneuvering

2.6 Experiments

In the previous section a quantitative comparison was made between the IMM estimator and a carefully tuned Kalman filter. From that comparison follows that the IMM estimator was a preferable choice for our application. To evaluate both the implemented detection component and the IMM estimator on the real set-up, a dataset has been gathered

from which two points will be discussed, the *localization performance* and the *tracking performance*.

2.6.1 Localization Performance

In Figure 2.13 the static mask of the field is depicted (in black), together with an overlay of two types of markers:

- *gray crosses*, indicating raw position measurements when the ball was localized,
- *black crosses*, indicating a measurement where the ball was lost.

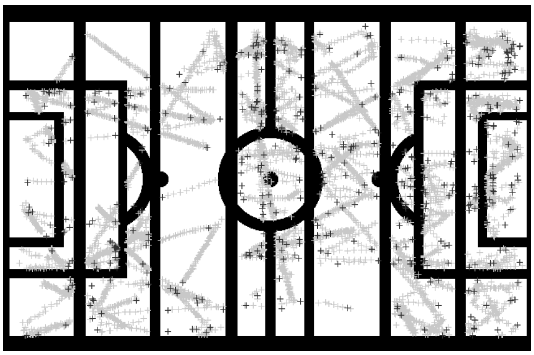


Figure 2.13: Detection results. Black crosses indicate a measurement where the ball was lost.

What can be observed is that the ball is frequently lost at the edges of the static mask. These are the positions where a large part of the ball is cut-off in the static masking operation [79]. However, the ball is also occasionally lost in between the masking contours. These events occur because of poor ball calibration, with respect to both the color segmentation classifier and the Circular Hough Transform parameter settings.

2.6.2 Tracking performance

The performance of the IMM filter was evaluated quantitatively in Section 2.5. Because on the real set-up it is difficult to obtain any form of ground truth on the actual ball position and velocity, another overlay is depicted. In this overlay, see Figure 2.14, the raw position measurements are now indicated in black crosses. The gray line indicates the position estimate from the IMM estimator.

From this picture, it can be observed that the IMM position estimates accurately correspond to the raw position measurements. Also, when the ball is lost the IMM filter

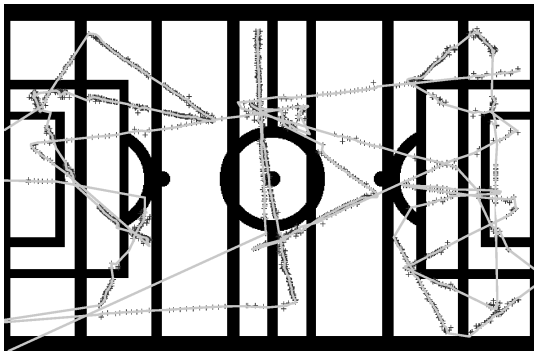


Figure 2.14: IMM performance. Black crosses indicate the raw measurements, the gray line indicates the IMM estimate.

adequately propagates the position estimate, maintaining a proper estimate on the ball's actual position.

2.7 Discussion And Conclusions

2.7.1 IMM versus standard Kalman filter

In Section 2.5, quantitative simulations were performed to investigate the performance of the IMM filter compared to a standard Kalman filter. From these simulations was concluded, that the IMM filter outperforms the Kalman filter based on the RMS values of

- the *position* error, which is required to intercept the ball accurately,
- the *velocity* error, which is required for an accurate heading reference.

What is important to note in this aspect, is that in these simulations the sample time was set at 0.005 [s] (200 [Hz]). In Section 2.3.4 was explained that the sample time is used to determine the 'target maneuvering index'. The larger this value, the more suited an IMM state estimator becomes for the application compared to a Kalman filter [89, 159]. If we consider that the sample time appears quadratic in the numerator of the maneuvering index, we can conclude that for lower frame rates (and thus larger sample times), the IMM estimator will outperform the Kalman filter even more than what we have shown now. As the expected goal of future robotic developments is to become smaller and therefore have less computing power, a lower frame-rate (and thus the required computational load) is possibly one of the future requirements of vision and estimation techniques. For this reason, will the IMM method become more attractive to be used in this specific application.

2.7.2 Ball Localization Performance

Although the ball was localized in the majority of the obtained measurements, improved calibration will lead to better detection results. Obtaining the best results from the localization component will be achieved by investigating the combination of color classification (leading to more positive ball pixels), edge filtering (obtaining more pixels that are on the outer edge) and circle detection. Especially tuning of the last component requires caution, since the chosen values for the radius and the accumulator threshold [101] depend heavily on the color and edge segmentation steps.

2.7.3 Ball Tracking Performance

A final conclusion is based on the results obtained from the experiments performed on the real set-up. Although the simulations in Section 2.5 have shown that the peak error for both the IMM and the Kalman filter techniques was no more than 12 mm, on the real set-up the actuated puppets on some occasions seem to miss the ball (i.e., a kick is executed too late). It can be assumed, that the main cause for this lies with the time delay between capturing an image and actuating the puppets. To determine the size of the maximum error caused by time delay, an analytical worst case value of 17.5 [ms] is determined in Appendix A, which is approximately equal to an empirically measured value of 10.6 [ms], see Figure 2.15.

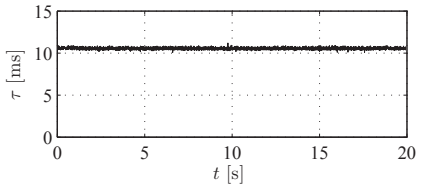


Figure 2.15: Measured time delay.

The maximum error that can be caused by this time delay occurs at maximum speed (5 [m/s]), and is therefore equal to 53 [mm]. This value explains why sometimes the puppets kick too late[4]. Solving for the time delay can be performed by *predicting* the current state estimate over the delayed time based on a constant velocity model. Further work has to reveal if this proposed solution will solve the problem.

[4]Remember that this error only occurs at maximum speed

Chapter 3

Policy Learning Using Greedy-GQ(λ)

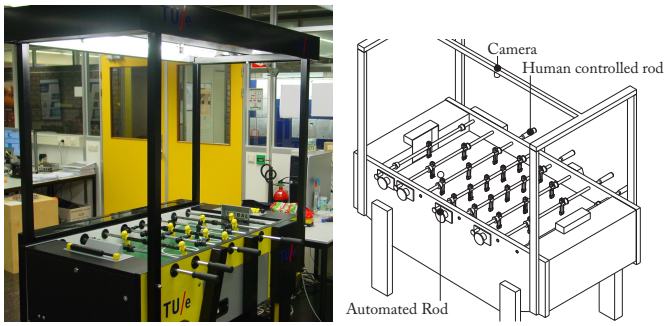
This chapter presents the design of a policy for a custom made, highly dynamic football table setup. As the design of an optimal policy for such a complex system is difficult to achieve manually, a recently developed Reinforcement Learning technique called Greedy-GQ(λ) *is adopted to serve as a design tool. To deal with the typical curse of dimensionality, function approximation and temporal action abstraction are applied to condense the large state and action spaces of this system, enabling the learning algorithm to be run under time and memory constraints on a modest everyday desktop PC. The integration of this work is experimentally validated, where the performance of Greedy-GQ(λ) is compared against well-known existing methods for learning. Additionally, a detailed simulation model has been designed to safely and efficiently perform experiments, and to learn an initial policy that can be applied on the real table. At the time of writing, this work can be considered as a first evaluation on the application of Greedy-GQ(λ) on this type of complex and adversarial system.*

This chapter is based on *"Policy Design For A Complex And Adversarial System Using Greedy-GQ(λ)"*, R. Janssen, E. Stoltenborg, G. Mohanarajah, R. van de Molengraft and M. Steinbuch, Machine Learning, 2014 (in preparation)

3.1 Introduction

The increasing research interests for coordinated satellite flight, multi-player computer games, process technology and collaborative service robots push the development of new techniques to operate high dimensional coordinated systems in dynamic environments.

Relevant research topics include large scale machine learning techniques, tracking and data association, control methods for multi-agent systems and *human-in-the-loop* robotic systems. To perform related experiments in a small and well-controlled environment, an automated soccer table has been developed, see Figure 3.1.



(a) (b)

Figure 3.1: Eindhoven University automated football table.

The specific purpose of this table is to serve as a small-scale testbed to validate methodologies of the above mentioned research topics.The table consists of an official *foosball* table, which has been equipped with an overhead camera to track the ball, and on one side with automated rods, allowing it to play against humans as a robotic adversary. The software for this system has been developed in MATLAB/Simulink [178], with the additional support for open-source libraries, such as Prosilica [5], OpenCV [40], and Armadillo [154]. A more detailed description of the design of the table can be found in [78], and the methods used to detect and track the ball are described in [80].

3.1.1 Designing a smarter control strategy

The current control strategy of the table entails a straightforward return of the ball when it approaches one of the mechanized puppets. A challenge therefore lies in the development of a smarter control strategy, for which manual modeling is often still the most widely applied approach. This approach has similarities with planning based approaches, such as A^* [66] or *Dynamic Programming* techniques [20], which base their decisions on an accurate model of system dynamics, the effects of the applied actions and the expected responses of the environment. As systems become more complex, non-linear and stochastic, accurate models are however more difficult to obtain by manual design. In stochastic games, such as computer *GO*, recent planning approaches involve forward-search using *upper-confidence bounds* on tree search [10] and Monte-Carlo Tree Search [35]. Such methods involve either a model of the environment, or an accurate black-box simulator that can be used for *real-time* projection, which are highly restrictive requirements to their practical application. Modeling a control strategy by manual design can also be very time consuming and often suboptimal, because anticipation, tuning and the incorporation of all possible scenarios is tedious and often not feasible. Also, due to

the static nature of these control strategies, changes in the environment which are unaccounted for at design time can cause degradation in the performance of the strategy. Furthermore, in adversarial environments with (human) opponents these strategies can become predictable and easy to counter.

For these reasons, it is desirable for a system to autonomously *generate* and *adapt* a strategy, based on the experienced response the system gains by interacting with the environment. A modern day method that is suitable for this task is *Reinforcement Learning* [166]. Reinforcement learning requires only the experience of interacting with the environment to generate a strategy, by basically using a trial and error approach. It observes transitions and rewards and derives a strategy, or so-called *policy*, from a learned input-output representation of the systems states and actions. This representation is typically called a *value function*, and represents the importance of a certain state or state-action pair with respect to a chosen policy and reward structure. It has been shown that policies generated by Reinforcement Learning often trump their hand-coded rivals [163]. Furthermore, recent advances in the field of Reinforcement learning have made it applicable to systems with large state-action spaces, by generalizing over states and clustering actions into condensed forms [17].

From the available existing on-line Reinforcement Learning algorithms, this work targets the (stochastic) gradient based Temporal Difference method Greedy-GQ(λ) because of its theoretical convergence properties, and because it scales linearly in computational complexity with the number of parameters. In this work, it will be compared to existing methods, such as *grid-based* Q(λ), which is guaranteed to converge and has low computational costs, and to *approximate* Q(λ). The latter only has convergence guarantees under very restrictive conditions, but often converges quicker and is computationally cheaper than gradient-based algorithms (such as Greedy-GQ(λ)). According to the developers of the algorithm, Greedy-GQ(λ) has not yet been studied when applied to a large scale real-world problem.

3.1.2 Problem statement

The concrete problem statement of this work is defined by the development of a more sophisticated control strategy for the automated football table, for which considering the system's complexity and adversarial character, an off-policy, gradient based Temporal Difference learning technique will be adopted as the preferred design tool. Within this field, Greedy-GQ(λ) will be selected as the proposed learning algorithm, as it claims superior convergence guarantees over comparable methods and moderate computational demands.

3.1.3 Contribution

Although experiments of Greedy-GQ(λ) have been conducted on a robotic multi-sensor platform called *Horde* [169], none of these experiments involved a large-scale and ad-

versarial application as the system described here. Furthermore, this work describes efficient, reusable implementations of function approximation and temporal action abstraction, hereby demonstrating how the application of such an algorithm can be applied to a complex system while still being able to execute under time and memory constraints on a modest everyday desktop PC.

3.1.4 Outline

The following section presents a general overview of the learning system, and roughly describes each of the involved components. The section thereafter discusses these components, and how the involved methods are selected and applied. After that, several experimental test cases will be described, in which the performance of Greedy-GQ(λ) is compared against well-known existing methods. This section will also describe the transferability issues that arise when a policy learned for a simulation model is applied to its real counterpart. Finally a conclusion of the described work is presented, together with a discussion on future work, proposed to improve upon the current implementations.

3.2 System overview

Learning is applied to the system based on the general learning architecture sketched in Figure 3.2.

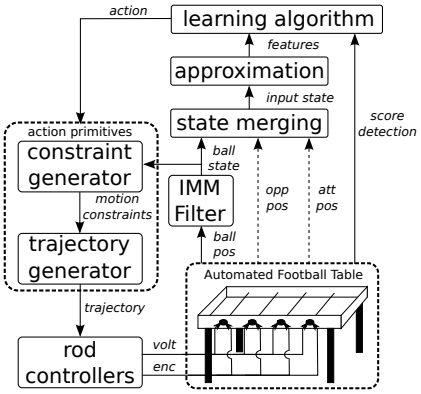


Figure 3.2: Learning architecture.

During learning, the learning algorithm (Section 3.3.2) receives rewards related to the goals scored. Based on these rewards and a selected behavior policy, the learning algorithm selects an action which is converted into an executable trajectory by the *action primitives* (Section 3.3.1).

Through the overhead camera the (raw) ball position is detected, from which a full ball state is reconstructed by an Interacting Multiple Model (IMM) filter. This filter will not be further addressed here, but a detailed description of its implementation on the football table can be found in [80].

Together with the position of the (human) opponent rod and the (mechanized) attacker rod, the ball state forms the *input state* to the learning algorithm. This work describes the use of an approximation to represent the input state efficiently, hereby transforming the representation of the input state into a smaller set of approximation *features*. This approximation will be discussed in Section 3.3.2. In the upcoming section, the design of the action primitives will be described firstly.

3.3 Application of methods

3.3.1 Design of action primitives

In various mechatronic applications, optimal or near optimal solutions to sub-tasks, so-called *action primitives*, can be derived from separate simplified learning procedures or fine tuning by hand. As indicated previously in Figure 3.2, this work describes the use of action primitives for translating (symbolic) discrete actions, available to the decision maker, into executable trajectories. The upcoming section presents the implementation details for a set of naturally occurring action primitives applicable to the presented type of system.

For the automated football table 5 distinct action primitives have been devised, see Figure 3.3.

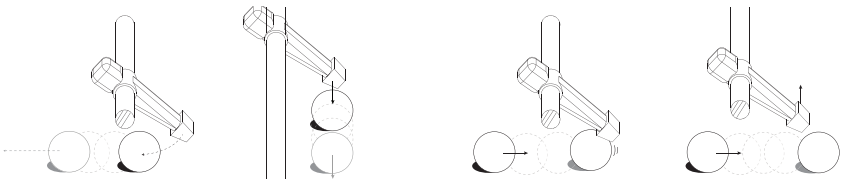


Figure 3.3: Available actions on the automated football table. Shoot (a), Tap left/right (b), Take (c) and Wait (d).

- **Shoot** performs a hard shot,
- **Tap left/right** taps the ball with the side of the puppet, to either left or right,
- **Take** intercepts and stops the ball with the front or back side of the puppet's feet,

- **Wait** explicit do-nothing (avoid if necessary)

These action primitives enable the transformation of a discrete action into an executable trajectory. Action primitives that represent a temporarily extended course of a discrete low level action are referred to as *temporally abstract actions* in the reinforcement learning literature [168].

Temporal action abstraction

In a continuous state space, a temporal action abstraction can be seen as the conversion of a discrete (often symbolic) action primitive into an executable reference signal, where an in-between step can be found in the generation of a set of motion constraints. This process is schematically depicted in Figure 3.4.

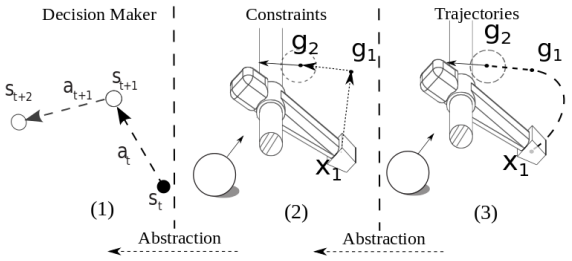


Figure 3.4: Temporal action abstraction: (1) The decision maker decides on an action primitive a_t at time t and state s_t. (2) State of the ball is extrapolated (dotted circle) to determine the intersection point. Based on the extrapolated intersection point and the selected action, constraints g1 and g2 are generated using heuristics. (3) Temporal trajectories are generated based on the current position and constraints.

These constraints can become noisy when they are based on features in the state $s \in S$ which are inaccurately measured, i.e., an inaccurate measurement of the ball or an opponent puppet. Moreover, if extrapolations are used for e.g., calculating the constraints of an interception task (such as a trying to predict where a mechanized puppet has to intercept the ball), the noise on these constraints will be amplified. A commonly used method for smooth motion generation based on noisy constraints is the use of a chain of quintic polynomials, often referred to as a *spline* [149], which involves the calculation of six parameters at each update of the constraint. Because of the unstable nature of a polynomial, updates also need to be performed after completion of an episode. Also, solving a set of six equations at every constraint update is a computationally expensive process and there is no way to pose intermediate constraints on e.g., maximum acceleration and velocity.

Another method to find parameters is by solving a convex optimization problem, which does allow such constraints. However, avoiding obstacles requires carefully selected via-points, and finding such points effectively is difficult. This is most readily achieved by adding positional constraints to the optimization problem [11, 157]. Solving such a problem however is computationally expensive, rendering the method not feasible for high-rate real-time systems, especially if frequent re-planning is required. More computationally efficient discrete methods like [98], use a decision tree to generate trajectories, allowing to embed all types of constraints on maximum jerk, acceleration and velocity. However, adding autonomous collision avoidance to the latter would still require significant computational effort as it would again involve solving e.g., an optimization problem.

Attractor dynamics for motion generation

Another method for constraint based motion generation that recently became more popular is the use of attractor dynamics. This method uses a known stable dynamic system to generate point to point trajectories, where the target point is chosen as the equilibrium center of the attractor, hereby guaranteeing stability and convergence. This work uses an approach based on Dynamical Movement Primitives (DMPs) [74], which use a canonical attractor augmented with an extra forcing term to model non-linear system dynamics. This allows the modeling of arbitrary complex trajectories, while still guaranteeing stability and convergence. The describing model is computationally efficient and allows straightforward obstacle avoidance through the extra forcing term.

A DMP for the described system can be described by a mass-spring-damper system given by

$$\tau \dot{x}_1 = x_2 \tag{3.1}$$

$$\tau \dot{x}_2 = \alpha(\beta(g - x_1) - x_2)) + f \tag{3.2}$$

Here g is the goal position, α and β are time constants and τ is a temporal scaling factor. The states x_1 and x_2 represent the reference position and velocity that the agent, a mechanically controlled puppet in this case, has to follow. The damping and spring terms, α and β respectively, are often chosen so that the system is critically damped ($\alpha = 4\beta$) which yields fast, monolithic convergence towards the goal. The larger α and β, the faster the convergence. Finally, f is the forcing term that is used to arbitrarily shape the trajectory. Although typically the forcing term f is a function of time, note that in this work f is regarded as a function of system state and used for obstacle avoidance.

Point-to-point motion generated using the original DMP formulation, suffers from large accelerations at the beginning of a motion due to the large error term $(g - x_1)$ used in (3.2). Although a solution to this is presented in [135], it comes with a significant increase in computational cost. Since this diminishes the computational advantage over a spline based approach, a more pragmatic solution is proposed that bounds the error term $(g - x_1)$. Governing this term can effectively pose a soft-constraint on maximum acceleration

and velocity, and is implemented by replacing g in (3.2) with g_b according to

$$g_b = \max(\min(g, g_{\max}), g_{\min}) \tag{3.3}$$

where

$$g_{\max} = \max(x_1 + x_2/\beta + \frac{a_{\max}}{\alpha\beta\tau^2})$$
$$g_{\min} = \min(x_1 + x_2/\beta - \frac{a_{\max}}{\alpha\beta\tau^2})$$

This is essentially the same as saturating $\dot{x}_2$ and does not danger convergence. However, if the maximum acceleration a_{max} is exceeded often convergence is no longer monolithic, and overshoot occurs.

As most constraints are based on extrapolation, they will typically be more noisy at the start of a motion due to extrapolation of an initial location to a target location. To incorporate this knowledge into the trajectory generation, the initial stiffness is modulated using a *Gompertz* sigmoid whose dynamics are given by

$$\dot{y} = \alpha_s \log(1/y)y \tag{3.4}$$

where y denotes the value of the sigmoid, and α_s denotes the stiffness or growth factor. The initial value y_0 is chosen close to zero and the stiffness is chosen such that $y(T) \approx 1$ and the end of the movement. Applying the Gompertz function and the saturation to the system described in Equations (3.1-3.2), results in the following new system equations given by

$$\tau\dot{x}_1 = x_2 \tag{3.5}$$
$$\tau\dot{x}_2 = y\alpha(\beta(g_b - x_1) - x_2)) + yf \tag{3.6}$$

The effect of the Gompertz function on the system is visualized in Figure 3.5.

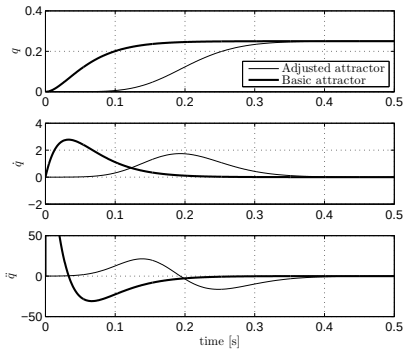


Figure 3.5: Basic and adjusted attractor dynamics.

Ball avoidance and non-zero target velocities

In addition to generating reference trajectories towards stationary targets, the devised action primitives are required to generate reference trajectories whose constraints include avoidance of the ball and non-zero final velocity targets.

For ball avoidance the forcing term f described in Equation (3.6) is constructed. For this an intuitive 2D-Gaussian potential field is used, which is a function of the agent's position x_1 and the ball position x_0, and is given by

$$f(x_1, x_o) = \frac{f_{\text{dir}}}{||f_{\text{dir}}||} K_a \zeta. \tag{3.7}$$

Here K_a is a scalar gain, and ζ is the Gaussian potential given by

$$\begin{aligned}
\zeta &= e^{-u\delta_{o,x}^2} e^{-2v\delta_{o,y}\delta_{o,x}} e^{-w\delta_{o,y}^2} \\
u &= \frac{\cos^2\theta}{2\sigma_x^2} + \frac{\sin^2\theta}{2\sigma_y^2}, \\
v &= -\frac{\sin 2\theta}{4\sigma_x^2} + \frac{\sin 2\theta}{4\sigma_y^2}, \\
w &= \frac{\cos^2\theta}{2\sigma_y^2} + \frac{\sin^2\theta}{2\sigma_x^2}, \\
\delta_o &= x_o - x_1
\end{aligned}$$

where θ is the rotational angle of the 2D-Gaussian potential field with respect to the x-axis. Finally, $f_{\text{dir}} / \parallel f_{\text{dir}} \parallel$ is a unit vector perpendicular to the vector δ_g directed towards the goal. See Figure 3.6 for a schematic representation of the above.

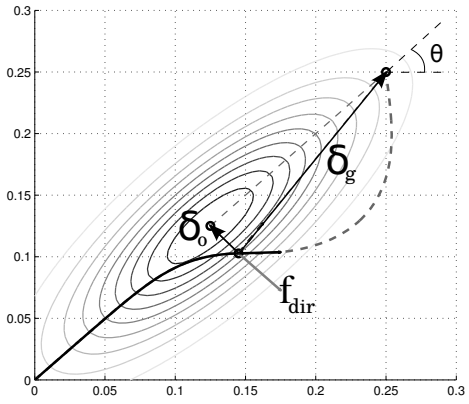


Figure 3.6: Ball avoidance with indication of directional vectors.

Desiring a certain velocity trough a target position, i.e., a constraint on the target velocity, can be achieved by defining a *moving target* [93], which basically makes the target goal g time/phase dependent. This results in an altered equation for $\dot{x}_2$

$$\tau\dot{x}_2 = y\alpha(\beta(g_m(t) - x_1) - x_2 + \dot{g}\tau)) + yf(x_1, x_o) \tag{3.8}$$

where $g_m(t)$ is simply chosen to be time dependent with constraints $g(T)$ and $\dot{g}(T)$, yielding $g_m(t) = g(T) - (T-t)\dot{g}$, $\dot{g} \to \dot{g}(T)$, where T denotes the target time. This basically yields a long 'run up' before approaching the target g, which for large velocities can yield a large difference $\delta_{gm}(t) = g_m(t) - g(T)$. To avoid this, saturation is applied to $\delta_{gm}(t)$, so that the difference can not become larger than a certain value. This results for $g_m(t)$ in

$$g_m(t) = g(T) + \max(\min(\delta_{gm}(t), \delta_{g_{max}}), \delta_{g_{min}})) \tag{3.9}$$

This 'run up' towards the goal g and the saturation of $g_m(t)$ are visualized in Figure 3.7, where $\delta_{g_{\min}} = -1$.

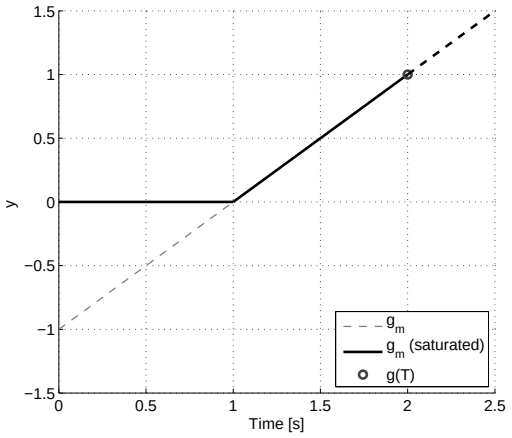


Figure 3.7: Goal at $y = 1$ with velocity $\dot{y} = 1$ at $T = 2$ and saturated $g_m(T)$.

In Figure 3.8, the trajectory for both a spline and altered attractor dynamic are shown for a desired velocity $\dot{g} = \begin{bmatrix} 0.5 & 0 \end{bmatrix}^T$, trough the goal position $g = \begin{bmatrix} 0.25 & 0.25 \end{bmatrix}^T$.

A more complex situation arises when the puppet is initially facing the ball, and it is required to hit the ball from behind. Here the puppet initially starts off with ball avoidance switched on, and subsequently turned off at a fixed time before the final target time. This fixed time interval is determined empirically. See Figure 3.9 for a schematic description of this situation.

For each of the described action primitives in Section 3.3.1, these constraint-based attractor dynamics are accessible in order to generate valid trajectories.

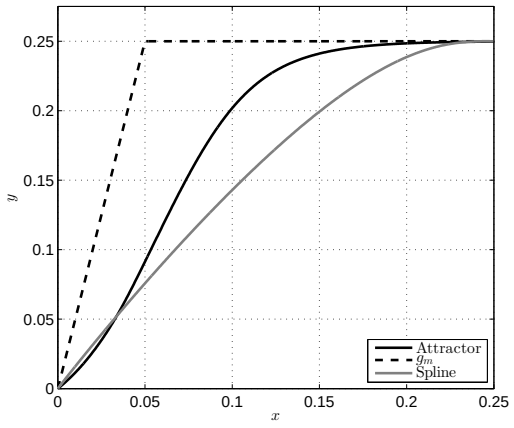


Figure 3.8: Attractor with moving target and spline-based trajectory for velocity constraint

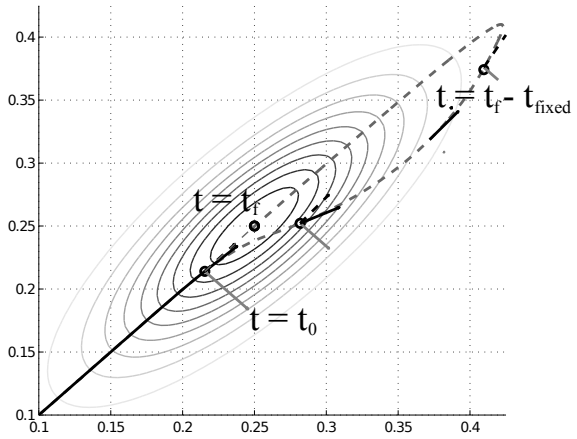


Figure 3.9: Avoid and Hit: At $t = t_0$ the puppet and the ball are moving towards each other. From $t = t_0$ to $t = t_f - t_{\text{fixed}}$ the collision avoidance is active, where t_f is the final impact time and t_{fixed} is a fixed time determined heuristically. After $t = t_f - t_{\text{fixed}}$ the collision is switched off and the puppet moves towards the ball to hit it from behind.

3.3.2 The learning algorithm: Greedy-GQ(λ)

This section presents Greedy-GQ(λ), the core learning algorithm. First a brief introduction of the involved Reinforcement Learning techniques is presented, after which the selection of Greedy-GQ(λ) is motivated for the described application. Optimization techniques and specific implementation details are presented at the end.

Reinforcement learning basic theory

In Reinforcement Learning, a learning agent selects an action $a_t \in A(s_t)$ in state $s_t \in S$, transits to a new state $s_{t+1} \in S$ and receives reward r_{t+1}. The goal of the agent, is to maximize the sum of obtained rewards over time, represented by the *value function*

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left\{\sum_{k=t}^{\infty} \gamma^k r_{k+1} | s_t = s\right\} \tag{3.10}$$

where $\gamma \in (0,1)$ is the discount factor and π is the given policy that describes the probability of selecting action a at state s.

The value function of Equation 3.10 can be used to *predict* the value of a given policy π. For *control* problems, such as presented here, an *action-value* representation can be used

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\right\} \\ &=: TQ^* \end{aligned}$$

where T is known as the *Bellman operator*.

The optimal policy π^* that maximizes the total sum of rewards can be found by solving the Bellman Optimality Equation given by

$$Q^*(s,a) = \mathbb{E}\left\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\right\}$$

For low dimensional problems with known transition and reward models the above optimality equation can be solved by straightforward Dynamic Programming techniques [20]. In the case of the automated soccer table these models are however not available and difficult to obtain. Secondly, the state-action dimension of the presented system is of a much larger scale. To indicate the number of possible $Q(s,a)$-values of this system; if the ball state (consisting of x,y-position and velocity) is discretized in 1 cm and 1 cm/s intervals respectively, and the number of actions is 5, the total amount of state-action pairs is 10^9. Storing these values in double format would take 3.8 GB of memory, already too much for a modest everyday PC to be stored in memory. Section 3.3.2 describes model free variations of learning the value function described in Equation (3.3.2) through experience. Dimensionality reduction techniques, required to deal with the large system dimensions, are presented in Section 3.3.2.

Q-learning

Two main techniques exist for model free value function learning: Monte Carlo methods and Temporal Difference learning. A Monte Carlo method completes a full episode,

and then updates the value function for all states (or actions). As opposed, Temporal Difference learning updates the value function at every time step.

For the presented system, an extension to Watkins' *Q-learning* [186] is selected, a Temporal Difference learning method for action value functions, as this method finds the optimal action values under *off-policy* training. Off-policy training enables a system to learn about a *target* policy (e.g., an optimal policy), while following a different *behavior* policy (e.g., exploratory).

The iterative update rule for Q-learning is given by

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha\Delta_t \tag{3.11}$$

where Δ_t is the *Temporal Difference error* given by

$$\Delta_t := r_{t+1} + \gamma \max_a Q_k(s_{t+1}, a) - Q_k(s_t, a_t) \tag{3.12}$$

To improve sample efficiency, *eligibility traces* [166] can be used. Eligibility traces allow the assignment of credit to a memory of recorded action values, hereby bridging the gap between the occurrence of events and the collection of data. Eligibility traces are applied through

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha\Delta_t e_t \tag{3.13}$$

where

$$e_t = \begin{cases} 1 + \gamma\lambda e_{t-1} & \text{if } a_{t+1} = a^* \\ \gamma\lambda e_{t-1} & \text{if } a_{t+1} \neq a^* \end{cases} \tag{3.14}$$

with $e_0 = 0$ and $\lambda \in (0,1)$. In the presented system ε-greedy is chosen as the behavior policy, which has an ε chance of selecting a random (exploring) action and a $1-\varepsilon$ chance of selecting a greedy (i.e., optimal) action according to $Q : a = \pi(s) = \max_a Q_k$. The above tabular form of Q-learning is guaranteed to converge to an optimal policy if the states are visited infinite times [166].

Function approximation

Making Reinforcement Learning more efficient for larger problems can be achieved by applying *function approximation*, which attempts to represent the value function in a general form. This reduces memory requirements, but it also allows the *generalization* over states that have not been visited. This work focuses on the use of *parametrized linear function approximation* with *Gaussian radial basis functions* [166], because of convergence guarantees and moderate computational cost [23]. Furthermore, because of the known distributions of the basis functions, only basis functions *local* to the current system state have to be updated, eliminating the requirement for a computationally costly

global update (see Section 3.3.2). Using parametrized linear function approximation, the Q-function is approximated by

$$Q(s_t, a_t) \approx Q_\theta(s_t, a_t) := \theta^\top \phi(s_t, a_t) \tag{3.15}$$

where θ is the parameter vector and $\phi(s_t, a_t) := (\phi_1(s_t, a_t), \cdots, \phi_m(s_t, a_t))$ denotes the feature vector, containing the state-action dependent values of the basis-functions $\phi_i(s_t, a_t) \in \mathbb{R}$, $i = 1, \ldots, n$. For a 1-dimensional state x these values are calculated using

$$\phi_i(x) = e^{\frac{||c_i - x||^2}{2\sigma_i^{r2}}} \tag{3.16}$$

where the choices of the centers c_i and widths σ_i^r of the basis functions are derived through an empirical evaluation, involving the RMS error of the state observation as a measure. In this evaluation it is assumed that the RMS error depends either directly or indirectly (through wrongfully executed actions) on the measurement noise in the *observed state*. Therefore, have the centers and widths of the basis functions been made dependent of this noise through a variable ratio. For the case of having only one action primitive (*Shoot*), and only one state (the position of the ball in lateral direction), the outcome of this evaluation is depicted in Figure 3.10, where the RMSE of the value estimation is depicted against the distance and width of the basis functions.

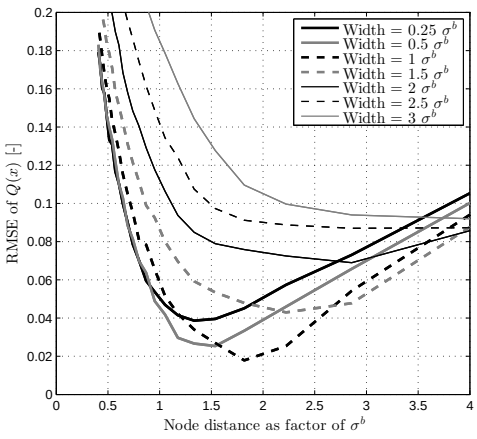


Figure 3.10: RMSE as a function of the RBF width and the RBF node distance expressed in the measurement noise of the observed ball state σ^b. The optimal point can be found at a width equal to σ^b and a node distance of $2\sigma^b$.

It can be seen that placing nodes further apart than $3\sigma^b$ yields a relatively large error, due to high generalization. On the other hand, choosing a node distance smaller than σ^b also yields a large error. This is due to a process called *over-fitting*.

Q-learning with function approximation

With the above function approximation, updating the parameter vector θ_t becomes core of the learning problem. Applying linear function approximation to *Q-learning* [128] results in a vector update rule similar to the grid-based Q-learning in Equation (3.11), and is given by

$$\theta_{t+1} = \theta_t + \alpha \delta_t \phi(s_t, a_t) \tag{3.17}$$

where the Temporal Difference error δ_t is given by

$$\delta_t = r_{t+1} + \gamma \max_{a'} \theta_t^\top \phi(s_{t+1}, a') - \theta_t^\top \phi(s_t, a) \tag{3.18}$$

The above Q-learning algorithm with linear function approximation played a vital role in modern Reinforcement Learning. Unfortunately, convergence for approximate Q-learning using the above updates has only been proven for very restrictive conditions. Most notably, Baird's counterexample [192] shows that Q-learning with value function approximation may diverge, making the approximation parameters go to infinity.

Greedy-GQ(λ)

Residual Gradient algorithms [13], propose a gradient descent based parameter update rule with respect to the mean squared Bellman error given by:

$$\| Q_\theta - TQ_\theta \|^2 \tag{3.19}$$

Unfortunately however, these methods induce a bias due to the double sampling problem [13].

In recent work, the *Greedy-GQ* update rule was presented in [116, 117]. This method also describes a gradient descent based parameter update rule, but now with respect to the mean square projected Bellman error (or MSPBE). Here, TQ_θ is first projected back to the parameter space, before the mean square Bellman error is calculated. A gradient correction term w is introduced, that serves as a gradient correction term and compensates for future Temporal Difference errors.

This addition result in the following two update rules

$$\theta_{t+1} = \theta_t + \alpha \left[\delta_t \phi_t - \gamma (\phi_t^\top w_t) \phi(s_{t+1}, a^*_{t+1}) \right]$$
$$w_{t+1} = w_t + \beta \left[\delta_t \phi_t - \gamma (\phi_t^\top w_t) \phi_t \right]$$

where $\phi_t := \phi(s_t, a_t)$ and

$$a^*_{t+1} = \arg\max_{a_{t+1}} (\theta_t \phi(s_{t+1}, a_{t+1})). \tag{3.20}$$

Pseudo-code of the Greedy-GQ(λ) implementation including the eligibility traces is given in Table 3.1 (from [117]).

Initialize w_0 to 0, and θ to arbitrarily
Choose small values for α, β and
set values for $\gamma \in [0,1]$
Repeat for each episode:
  initialize $e = 0$
  choose actions a_t, in state s_t according to $\pi_b(s_t)$
  observe sample, (s_t, r_{t+1}, s_{t+1}) and calculate
  the feature vectors $\phi_t(s_t, a_t)$ and $\phi_t(s_{t+1}, a^*_{t+1})$
  for each sample **do**
    $\delta_t \leftarrow r_{t+1} + \gamma \max_a \left(\theta_t^\top \phi(s_{t+1}, a)\right) - \theta_t^\top \ \phi(s_t, a_t)$
    if $a_t = a^*_t$, then $\rho_t = \frac{1}{\pi_b(a^*_t|s_t)}$; otherwise $\rho_t \leftarrow 0$
    $e_t = \phi_t + \rho_t \lambda e_{t-1}$
    $\theta_{t+1} \leftarrow \theta_t + \alpha \left[\delta_t e_t - \gamma(1-\lambda)(e_t^T w_t)\phi(s_{t+1}, a^*_{t+1})\right]$
    $w_{t+1} \leftarrow w_t + \beta \left[\delta_t e_t - (\phi_t^T w_t)\phi_t\right]$
  end for

Table 3.1: Greedy-GQ(λ) pseudo-code.

Parameter β denotes the step-size, which is similar to the learning rate α, and indicates the amount of correction that is being applied to the weight vector w with respect to the Temporal Difference error δ_t. The term ρ_t denotes the ratio between the probability of selecting action a by the target policy $\pi_t(a|s)$ and the probability of selecting the same action by the behavior policy $\pi_b(a|s)$. Since $\pi_t(a^*|s) = 1$ for Greedy-GQ(λ) , this implies $\rho_{a_t=a^*} = \frac{1}{\pi_b(a^*_t|s_t)}$, and $\rho_{a_t \neq a^*} = 0$.

Greedy-GQ(λ) can be considered as an effective learning method, with linear complexity in the number of approximation parameters, learning convergence guarantees, and straightforward integration with function approximation and eligibility traces. Because it is a Temporal Difference learning method, it allows continuous policy updates, keeping per-step calculation times constant and maintainable. Furthermore, because it allows off-policy learning, it enables the possibility to calculate the optimal policy, while still exploring for new opportunities. These reasons make it well suitable for the presented type of application; a large-scale, real-world system with an adversarial character.

Efficient computations through sparse updates

This section presents a technique to further reduce system dimensionality, with the purpose of requiring less computational and storage demands. The basic idea of this technique is that in the presented setup only a certain amount of local basis functions is required to represent a particular state [142]. This enables the updating of only the weights/parameters corresponding to these local basis functions and leave the others untouched, hereby gaining a huge computational advantage.

An example of these sparse updates is depicted in Figure 3.11, where both the neighboring (i.e., local) basis functions as well as the non-contributing basis functions are indicated.

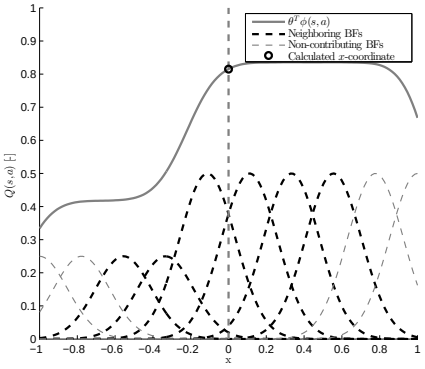


Figure 3.11: Local and neighboring basis functions. x indicates the distance between the basis function center and the position of the evaluated state.

Using a k-nearest-neighbor approach, an n-dimensional problem can be solved using only k^n nodes, without significant loss of accuracy. The approximate values for the relevant vectors are stored in ϕ_t^S, w_t^S, $\theta_t^S \in \mathbb{R}^{1\times k^n}$, together with an *index-vector* K_t^S used to transform back to the full problem, where $K^S(i)$ gives the index j in the real vectors $\phi_t(j)$, $w_t(j)$, $\theta_t(j)$ of the sparse vector element i, $\theta_t^S(i) : K_t^S(i) \rightarrow \theta_t(j)$. By calculating only local basis functions, the computational cost goes down significantly. In Table 3.2, the computation time of sparse calculation using 4 action primitives is compared to performing updates on the full problem [1].

Dimensions $\rightarrow$	2	3	4
Number of nodes $\rightarrow$	900	27e3	36e4
Full GQ-update	64ms	2.6s	41.9s
Sparse GQ-update(k=10)	2ms	21ms	0.446s
Sparse GQ-update(k=7)	0ms*	4ms	0.1s

Table 3.2: Calculation times of Greedy-GQ(λ) updates with 4 action primitives.

At $k = 3$, the first node neglected is at a distance of $3\sigma^r$, which corresponds to a feature value of the neglected node i of $\phi_i(x) \leq 0.011$. This results in an error of approximately two percent per state in the feature vector, which in respect to the gain in computational cost can be regarded as negligible.

[1] Calculations marked with * took less tan 1 μs. All calculations are the average of 100 calculations

3.4 Experiments

To test and compare convergence and performance, several distinct test cases have been defined. In these tests, Greedy-GQ(λ) is compared against grid-based Q(λ), approximate Q(λ) and a hand-coded policy. In addition to real experiments, also simulated experiments have been performed with a physics engine based simulator that closely resembles the real system. These simulations allowed to have several controlled experiments for analyzing various properties of the learning algorithms. Since the real set up is already covered in Section 3.2, this section starts with an overview of the simulation setup.

3.4.1 Simulator

Because typically a lot of episodes are required for learning algorithms to converge, a simulation model has been developed specifically for this purpose. Learning takes place in a simulator based on Gazebo [95] (Figure 3.12), which uses physics based on the Open Dynamics Engine (ODE) [162].

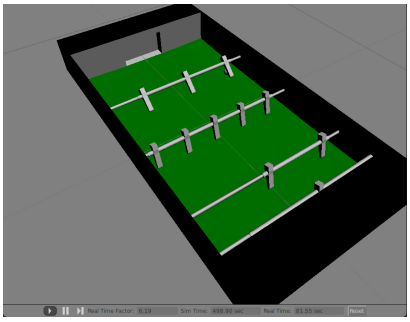


Figure 3.12: Gazebo simulation model.

For the purpose of this work, a plug-in was built that enables fast synchronized interprocess communication between Matlab/Simulink and Gazebo using shared memory. This plug-in allows us to run simulations with the real motion controllers. Furthermore, the ball detection algorithm is replaced by the ball-coordinates from the simulator, to which noise has been added to simulate the noise characteristics of the real detection algorithm. Based upon this, the Interacting Multiple Model (IMM) tracking algorithm [80] that estimates the state of the ball is also running in simulation, hereby enabling the simulator to mimic the real setup as much as possible. In analogy with Figure 3.2 from Section 3.2, a human-like opponent has been added in simulation to imitate the behavior of an average skilled player. To model this behavior, a straightforward mass-spring-damper is used, with an equilibrium point at the lateral position of the ball (so the opponent moves sideways with the ball, hereby trying to block the shot of the mechanized attacker). Fur-

thermore, when a goal is scored this is detected in the simulator and fed into the learning algorithm as an input to the reward structure.

3.4.2 Test cases

For the evaluation of Greedy-GQ(λ), 7 test cases are proposed, from which 6 are performed in the simulator. The 6^{th} test case will describe a scenario that is also transferable to the real table. Transferability and performance on the real table are therefore discussed in the 7^{th} test case. The concrete 7 test cases are:

1. a 2-dimensional (2D) setting in which only the lateral position and velocity of the ball (y and $\dot{y}$, see Figure 3.13) are available in the input state. This test case is performed without noise on the ball measurement, resulting in a perfectly estimated ball state by the IMM filter and hence, a perfect input state to the learning algorithm,
2. same as test case 1, but now with noise on the input state. This results in poorer ball state estimates provided by the IMM filter,
3. same as test case 2, but with the presence of (simulated) opponents (the goalie and the last line of defenders, see Figure 3.13 for puppet topology). In this test case, the state of the opposition is not available in the input state. Also, the opponents will exhibit a static behavior with respect to the ball, i.e., they will laterally follow the ball according to the opponent model discussed in Section 3.4.1,
4. same as test case 3, but now the opponents suddenly change their behavior during the game. They will switch from laterally following the ball to remaining on a static position, through which the algorithms adaptive characteristics will be evaluated,
5. same as test case 3, except that now the state of the opposition (lateral position of the rods holding the goalie and the last line of defenders) is available in the input state. This results in a 4-dimensional (4D) input state $[y, \dot{y}, y_g, y_{def}]$,
6. a 5-dimensional (5D) setting in which the opposition is present and moving as in test case 3, but again not available in the input state. Instead, the longitudinal position and speed of the ball (x and $\dot{x}$) and the angle of the attacking puppet θ_p are added. This results in the input state $[x, \dot{x}, y, \dot{y}, \theta_p]$,
7. an experiment on the real table, in which the policy learned in test case 6 is applied to the real system.

Reward structure and episode termination

Defining rewards for a Reinforcement Learning problem is typically an intuitive and empirical approach. For the automated football table the rewards are defined as follows:

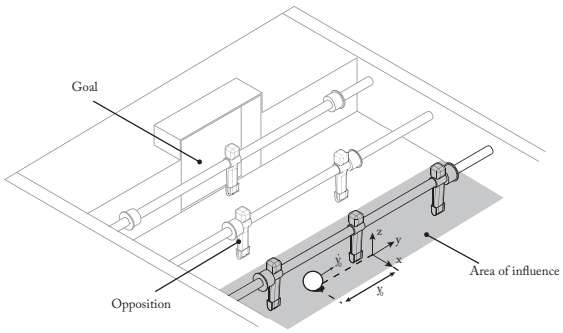


Figure 3.13: Coordinate system, puppet topology and ball position spawning line y_0 for the 2D setting.

- loosing the ball (such that it rolls out of the puppets reach, or *area of influence*, see Figure 3.13) without scoring a goal: **-2.0**
- running into a time-out (*e.g.* when the ball stopped moving at an unreachable position and a time-limit is reached): **-2.0**
- scoring a goal: **+4.0**
- applying an action: **-0.1**

Episodes terminate when one of the first 3 cases occurs.

Case 1: 2D input state without noise

In the first test case, Greedy-GQ(λ) is compared to grid-based Q(λ), approximate Q(λ) and a hand-coded policy in a 2D setting (ball y and $\dot{y}$), as depicted in Figure 3.13. The spawning states of the ball y_0 and $\dot{y}_0$ are drawn from a random distribution. The agent's goal is to score as quickly as possible, using the actions *Shoot*, *Tap left/right* and *Wait* [2].

In this test case, grid-based Q(λ) uses a moderately coarse tabular representation of $N_y \times N_{\dot{y}} \times N_a = 61 \times 21 \times 4$, where N_y and $N_{\dot{y}}$ denote the number of tilings in y and $\dot{y}$ respectively and N_a denotes the number of available actions. Function approximation used in Greedy-GQ(λ) and approximate Q(λ) uses a total of 20×10 (non-normalized) Gaussian RBFs, yielding a weight vector θ_t and w_t of length $20 \times 10 \times N_a$, where N_a is 4 in this case.

The added measurement noise on the ball detection σ^b is set to zero, resulting in perfect state estimates by the IMM filter. However, because the real motion controllers are used

[2] *Take* is here not relevant, as the ball state in this experiment is supposed not to vary in the x-direction.

to control the rods, tracking of the prescribed trajectories is imperfect and posed constraints might not be satisfied (yielding a stochastic action outcome). The experimental results are shown in Figure 3.14.

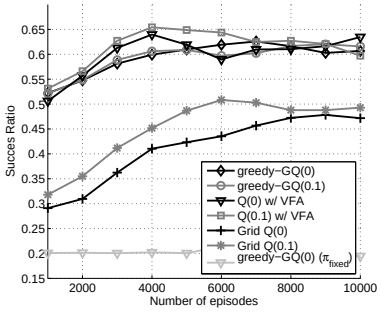


Figure 3.14: Learning performance moving average ($\alpha = 0.2$, $\varepsilon = 0.25$) for test case 1.

From Figure 3.14 it shows that Greedy-GQ(λ) and approximate Q(λ) converge far quicker than grid-based Q(λ), which is primarily due to added generalization from the use of function approximation. Also, approximate Q(λ) converges faster than Greedy-GQ(λ), despite using the same learning rate α. This is due to the gradient correction term w, which adjusts the updates with respect to the gradient, limiting the rate of change in the parameters. The trajectory of subsequent updates changes direction more slowly because of this, which causes slower convergence [50].

The large swing in performance, which shows in this case for approximate Q(λ), is likely due to variance in action outcome. This means that if a (supposedly optimal) action consecutively fails, the ε-greedy policy will start to give preference to another (possibly suboptimal) action, resulting in a change of policy. This causes these 'long term' changes in performance, which are not necessarily unique to approximate Q-learning. Using a lower, or variable α will limit this phenomenon. Figure 3.15 shows the policies learned by the Greedy-GQ(0) against the two dimensional lateral state (y, $\dot{y}$) of the ball.

Along the line $y = \dot{y}$ the ball is in a state moving towards the center. In this region Greedy-GQ(0) decides to shoot for the goal. Note that due to the temporal action abstraction shoot at a given state does not mean that the ball is intercepted at that state. In regions around $y = \dot{y}$ shoot intercepts the ball around the center $y = 0$. Another, interesting observation is the region around the line $\dot{y} = 0$. Here, when the ball is on the left ($y > 0$) the Greedy-GQ(0) taps right and when the ball is on the right taps left. But, when the ball is closer to the wall it taps towards the wall to get the ball to the center.

In order to compare the performance of the learned policies to the hand-coded one, the full-greedy ($\varepsilon = 0$) performance of the learned policies is used after 10^4 episodes[3]. The results are plotted in Figure 3.16, and the average greedy performance can be found in

[3]For this specific test case it takes approximately 1 hour of simulation time to complete 10^4 episodes.

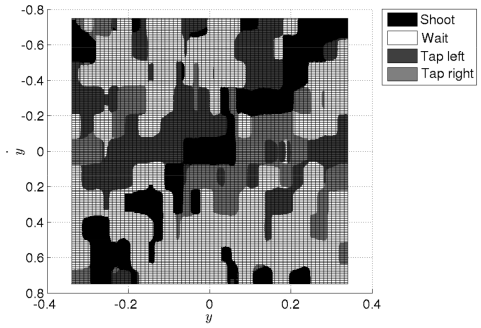


Figure 3.15: Learned Greedy-GQ(0) policy.

Table 3.3.

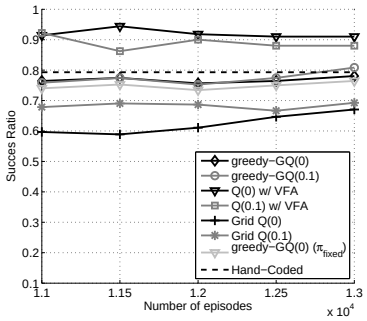


Figure 3.16: Greedy performance of learned policies and the hand-coded policy.

From Figure 3.16 can be derived that grid-based Q(0) is still improving, which indicates that this method, as opposed to the methods using function approximation, has not yet reached convergence after 10^4 episodes. This observation can be attributed to the more generalized representation of the value function, enabling faster convergence.

As indicated in Figure 3.14, the behavior policy π_b for Greedy-GQ(λ) is not chosen to be fixed, but instead an ε-greedy policy is used. When coupled with a low value of α, this should not danger convergence [116]. The result when using a fixed (stochastic) behavior policy, π_{fixed} is also shown in 3.14. Although this policy obviously does not perform well during learning, the learned policy shows comparable (greedy) results to that without the fixed behavior policy (see Figure 3.16).

The use of *eligibility traces* with grid-based Q(0.1) offers significantly faster convergence over the grid-based Q(0) update as sample efficiency is improved. However, it

does impose additional computational demands as more samples have to be updated. For Greedy-GQ(λ) the use of eligibility traces shows no significant difference in overall performance and convergence, and for approximate Q(λ) it yields a minor improvement. This minor (or insignificant) increase in performance can be explained by the fact that convergence is already fast due to the use of function approximation, yielding shorter episodes and smaller TD-errors. Moreover, because exploratory actions are cut-off in the trace (by $\rho \rightarrow 0$, see Section 3.3.2), the added effect of eligibility traces is further reduced. The hand-coded policy performs comparatively well (see Table 3.3), which for this problem is fairly simple to design. After 10^4 episodes, the grid-based Q(λ) policies perform significantly worse than the hand-coded policy, whereas the greedy performance of Greedy-GQ(λ) based policies are comparable. The policies derived using approximate Q(λ) outperform the hand-coded policy.

Case 2: 2D input state with noise

Next, the previous test-case is performed again, but now with noise in the ball measurement, affecting both the decision making process and performance of the primitive actions. In Figure 3.17 the learning performance is depicted.

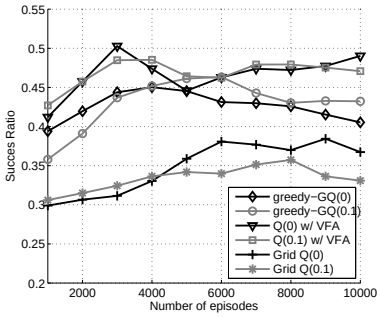


Figure 3.17: Learning performance ($\alpha = 0.2$, $\varepsilon = 0.25$) with noisy state estimations.

The greedy performance can be found in Table 3.3 and is depicted in Figure 3.18.

Again the policies generated by algorithms using function approximation perform better than their grid-based counterparts. Again, approximate Q(λ) converges the fastest. The added noise naturally causes more variance in the learning performance of all variants, which can be lowered by using an even smaller learning rate α. Notably Greedy-GQ(0.1) now shows a large swing in performance, which was also discussed in the previous test-case.

The use of eligibility traces yields a significantly better performance for Greedy-GQ(λ), indicating that enhanced sample-efficiency plays a more significant role in a noisy environment where generally more samples are needed. However, approximate Q(λ) again

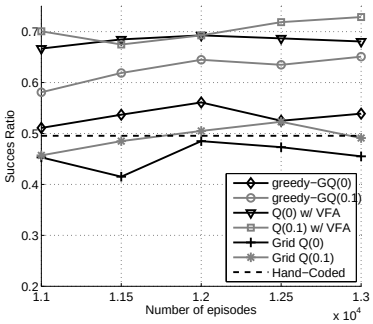


Figure 3.18: Greedy performance of learned and hand-coded policies with noisy states.

still only shows a minor improvement with the addition of eligibility traces.

Figure 3.18 shows that the hand-coded policy is performing significantly worse when noise is added. This is due to the fact that no adjustments are made to cope with the changed/worsened performance of the selectable actions. Such adjustments would require extensive manual tuning of this policy. Learned policies do constantly make adjustments, hence their degradation of performance due to the introduction of noise is smaller. As a consequence, the grid-based Q(λ) policies now perform on par with their hand-coded counterpart. Greedy-GQ(λ) and approximate Q(λ) both perform significantly better compared to the hand-coded policy.

	No Noise	Noise	Noise & Opposition
Greedy-GQ(0)	77%	53%	54%
Greedy-GQ(0.1)	78%	64%	63%
Q(0) (w/ VFA)	88%	68%	-
Q(0.1) (w/ VFA)	91%	72%	-
Q(0) (grid)	64%	50%	34%
Q(0.1) (grid)	68%	46%	37%
Hand-coded	79%	49%	37%

Table 3.3: Greedy performance in 2D test-case. (Q(λ) has not been evaluated in the last column, see Section 3.4.2.)

Case 3: 2D input state with unknown, moving opponents

Next performance is evaluated with (simulated) unknown opposition present, which is depicted in Figure 3.19. The greedy performance can be found in Table 3.3. This case was not performed for approximate Q(λ), as the trend of best convergence results shown in earlier test-cases is expected to persist in this test-case.

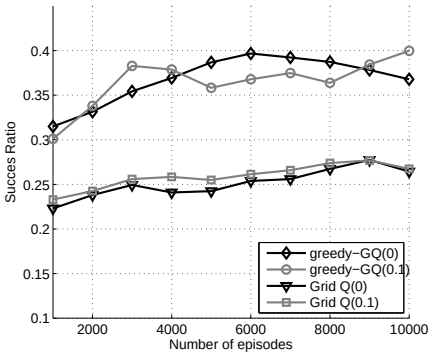


Figure 3.19: Moving average learning performance ($\alpha = 0.2$, $\varepsilon = 0.25$) with noisy state estimations and opposition present.

As mentioned, the overall learning performance again shows the same trend as in earlier test cases, however the gap in performance between Greedy-GQ(λ) and grid-based $Q(\lambda)$ grows compared to the last example. Greedy-GQ(λ) generates the best performing policy and grid-based $Q(\lambda)$ performs comparable to the hand-coded policy (see Table 3.3). Because the opponent shows consistent behavior it allows the agent to use its experience to improve upon its performance.

Case 4: 2D input state with unknown, moving/static opponents

In the aforementioned test-cases the environment was modeled to be static. In this test case, the environment will change during game play. The goal of this test case, is to investigate the algorithms adaptive capabilities. More formally said; the ability to adapt its target policy based on new experience.

In this test case, the learning agent has access to the (full) ball state, but the position of the opponent remains unknown. Half-way trough the game (around episode 2500), the opponents changes their behavior of laterally following the ball, into remaining at a static position. The result of this test-case can be found in 3.20.

What can be seen in the above figure, is that around episode 2500 something changes drastically in the system (in this case, a change in opponent behavior). Within 1000 episodes, the system has recovered its performance by updating its target policy to a new optimum.

What clearly has to be noted in this test case, is that although adaptivity is shown through policy iteration, it does not show the algorithm's capability for *on-line* (i.e., intra-episode) policy improvement. Unfortunately, the current simulator has not yet been developed to support this evaluation. For the presented system however, adaptivity through policy

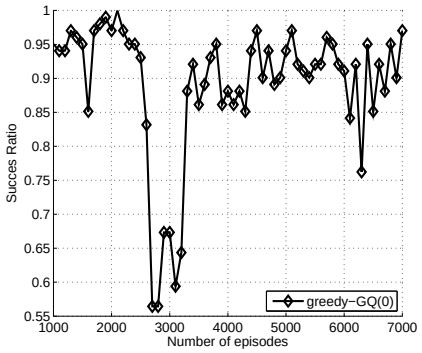


Figure 3.20: Greedy performance of learning agent with change in opposition behavior. (A performance of 1 was achieved because of a static ball placement, hereby greatly simplifying the learning problem.)

iteration is sufficient to cope with possible changes in system dynamics or opponent behavior, as the episodes involved in a game of table football are typically short.

Case 5: 4D input state with known opponents

Following is a test case where opposition is present, laterally following the ball, and their position is known to the learning algorithm. This setting results in the input state $[y, \dot{y}, y_g, y_{def}]$, where y_g and y_{def} are the lateral position of the goalie and the defender rod respectively. From this test case, the effect of increasing the dimensionality of the problem can be derived. The episodes are terminated in the same way as in Case 1-3. The performance during learning is shown in Figure 3.21, where the performance for the 2D case is compared to that of the 4D case.

Convergence is notably slower for the 4D case, which is to be expected as experience gets 'distributed' over a larger number of nodes. Also, variance in the solution is lower for the 4D counterpart. This can be attributed to the earlier statement that experience gets distributed. It is however also due to the fact that the opponent now no longer can be viewed as a factor of 'noise', i.e., the decision maker now can correlate the opponents position to the observed rewards. The 4D case is not applied to grid-based $Q(\lambda)$ because of excessive memory requirements.

The hand-coded policy was altered to use the position of the opposition as well. The adapted policy scores 46% of the time in the presence of noise and opposition, clearly benefiting from the added input.

The 4D based Greedy-GQ(0.1) policy achieves a greedy performance of 50% after 10^4 trials, whereas the 2D based Greedy-GQ(0.1) policy scores 62% without knowledge of the opposition. To achieve a similar performance, the 4D learning agent would require

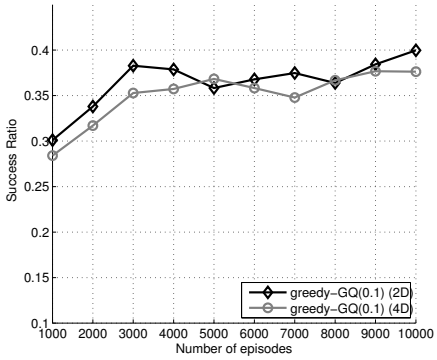


Figure 3.21: Moving average learning performance ($\alpha = 0.2$, $\varepsilon = 0.25$) with a 2D versus a 4D input state.

a lot more trials. Because of the used opponent model, the ball state effectively contains information about the opponent. So if the 4D policy would eventually perform better is difficult to predict. In a scenario where the opponents change their behavior however, adding these states will show a bigger benefit.

Case 6: 5D input state with longitudinal ball states and attacker angle

The final test-case uses all primitive actions (*take* is now available), where x_0, y_0, $\dot{x}_0$ and $\dot{y}_0$ are drawn from a random distribution (see Figure 3.22).

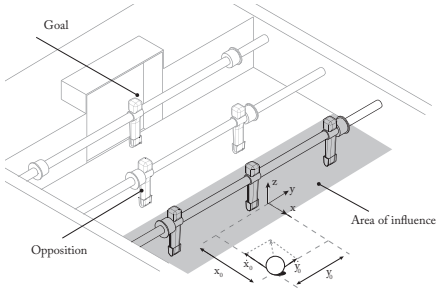


Figure 3.22: Coordinate system and ball spawning positions for the 5D case.

Here only results for Greedy-GQ(0.1) are shown, as differences in performance are now sufficiently clear. The input used here is 5-dimensional: $\begin{bmatrix} x & y & \dot{x} & \dot{y} & \theta_p \end{bmatrix}$, where θ_p is the controlled puppets angular position. The input state is approximated using $N_{rbf} = 31 \times 21 \times 11 \times 11 \times 7$ RBFs, yielding a vector θ and w of length $N_{rbf} \times 5$. Updates are done using $k = 3$ nearest-neighbors per dimension, so sparse vectors of length 3^5 are

updated. Convergence of Greedy-GQ(0.1) is shown in Figure 3.23.

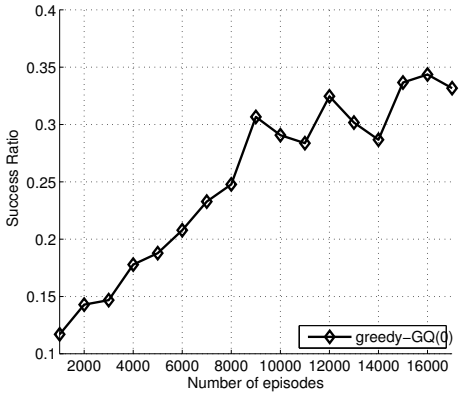


Figure 3.23: Learning performance for 4-dimensional problem

The final greedy performance in this test-case is 67% , which is comparably high considering the increased size of the state space. This is mostly due the fact that after successfully performing the action *Take*, the problem becomes very similar to the 2D case.

For this problem the initial performance is lower. This is due to the fact that the selection of the first action of the trial is far more important, e.g., avoiding the ball at the start of a trial will just result in termination, which does not occur in the 2D case. This indicates that less frequent exploration, or *Soft-Max* action selection [166] instead of the epsilon-Greedy approach, might be more suitable for this problem. The latter was already applied to Greedy-GQ, called Soft-Max-GQ, in [48].

The hand-coded policy is altered again to suit the problem, where the performance of the hand-coded policy is only 26%. The policy clearly needs tuning, making a comparison in this situation unfair.

Case 7: On the real setup

In the final test case, the previous learned policy is applied on the real set-up, in order to test transferability. Actual learning, albeit computationally feasible, is not applied here because setting up repetitive trials is a highly time consuming operation.

Instead, the learned policy from test case 6 is executed inside the systems real-time control loop, where an optimal action is calculated on-demand, using the sparse calculations mentioned earlier[4]. Using sparse calculations makes it possible to finish these operations

[4]It would be more effective to perform these calculations in advance.

when running at 2000 Hz in real-time [5], while simultaneously running the computationally demanding ball detection and state estimation algorithms. This indicates that learning is also feasible in this setting, as adding the update rules require only moderately more computational effort.

In Figure 3.24, key-frames are shown of an attacking move being played during a successful execution of the learned policy:

- (a) The ball is approaching from the rear,
- (b) The policy selects the action *Take*,
- (c) The action *Take* results in a successful stop of the ball
- (d) The policy selects the action *Tap-left*, where initially the ball is avoided until the puppet is at the right side of the ball and is capable of tapping it to the left,
- (e) The policy selects the action *Shoot*,
- (f) A goal is scored.

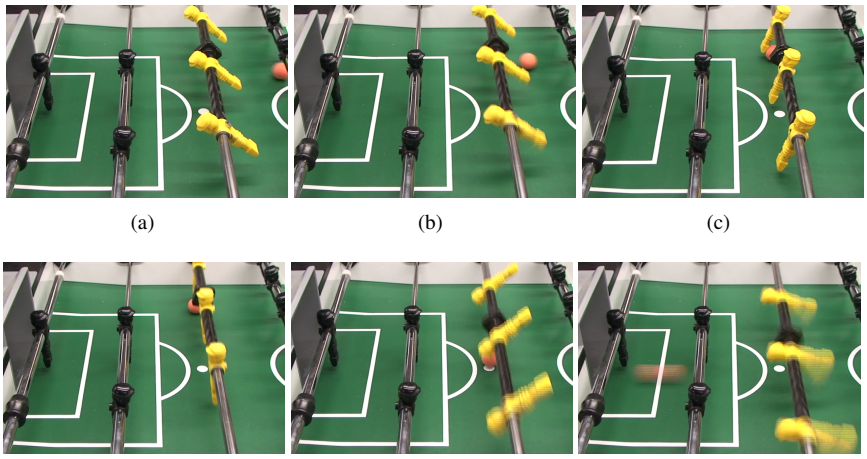


Figure 3.24: Experiments on the real setup.

Although the learned policy performs well, success rates comparable to those achieved in the simulator (67%) are not achieved. For example, when handling the ball frequent occlusions hinder accurate and consistent detection (and because of that, good action execution). Although the IMM filter tries to handle these occlusions, the estimated ball state is never as accurate as in the simulator. Also other issues like an imperfectly round ball, and a not perfectly leveled playing field also add to more failures.

[5] Performed on a 3 GHz dual-core machine, with 3.8 GB of memory available.

Besides these additional factors of noise, transferability largely hinges on an accurate simulator. Any inaccuracies in the simulation model can cause outcomes of actions to significantly differ from reality. This could affect the decision making, because e.g., a larger rolling friction of the ball will cause it to roll less far during execution of the action *Shoot*. Considering the action takes a finite amount of time, the displacement of the ball during the action (before collision occurs) is now significantly different, causing it to miss the goal. To cope with this, more thorough tuning of the simulator is required. These pitfalls mostly affect the action *Shoot*, as e.g., *Tap-left* or *Tap-right* normally don't result in a terminal state (i.e., not losing possession). Instead, the policy re-selects the action in order to get in front of the goal.

3.4.3 Discussion

Although not discussed here and in most other literature, designing the learning environment for a complex system can be a non-trivial task. Attributing rewards to the correct action may prove difficult, as they are often delayed because of non-instantaneous state transitions. Hence, careful design of episode termination and the assignment of rewards is highly important. RL however does have to power to overcome certain problems like these delayed rewards, because the reward still affects previous actions and hence indirectly increases the value of the actual 'responsible' action .

Designing a reward function however is often far more easy than designing a policy itself, as only the goal has to be represented. This does not take away from the fact that great care must be taken when designing the rewards, as imbalanced rewards might make the agent behave undesirably. Once the learning environment is in place however, development of a policy is quick and tuning is no longer necessary. Even when a discrete set of actions is presented, the learning agent can still devise solutions that were not anticipated for. For instance, in simulation the agent often used the wall to bounce the ball off, followed directly by a shot resulting in a goal.

This work showed that, in a more simple environment, hand-coded policies will perform very well. For these cases setting up an RL experiment will generally cost more time than creating an effective hand-coded policy. But even here, learned policies can eventually lead to comparable or better performance. When other factors are introduced that are unaccounted for in the hand-coded policy, such as noise and unknown opponents, the learned policies started to outperform their hand-coded rivals. This is simply due to the fact that, trough experience, the learned policies over time will achieve optimal behavior.

Greedy-GQ(λ) showed performance close to approximate Q(λ), with the added benefit that in combination of a fixed, or slowly changing, behavior policy convergence can be guaranteed. Greedy-GQ(λ) however, requires twice as much memory and doubles the computational demand. Moreover, situations in which convergence issues arise are rare in reality. Hence, for most cases, like the ones presented here, using approximate Q(λ) should suffice.

The use of *function-approximation* proved highly useful in large states-spaces. Mostly

because of the additional generalization achieved, but it also makes the use of a high-dimensional state-space more feasible. However, memory demands will still remain large for high-dimensional problems, and performing updates to the full-problem is highly computationally demanding. Using spare updates relieves this computational demand, although it nevertheless grows exponentially with respect to the dimensionality of the state-space. Also, convergence will often be slower for a higher dimensional problem. Therefore, the input to a learning system has to be carefully selected.

Applying *temporal action abstraction* greatly reduces the complexity of the problem, making the application of RL on the presented system feasible. However, it should be noted that creating a robust and flexible set of actions is a time consuming task. Although not discussed here, deciding on the granularity of the actions is also an important design consideration. A *coarse* granularity would achieve a higher level of abstraction and hence faster convergence, whereas a *fine* granularity would allow more flexibility and a possibly better performance.

Applying *eligibility traces* for both methods only provides a marginal improvement in speed of convergence. In the presented experiments however, they did not justify the increased computational effort.

3.5 Conclusion & Future work

This chapter presents the application and evaluation of the first off-policy, linearly complex, gradient based TD learning algorithm, *Greedy-GQ(λ)*, on a real-world mechanical set up.

While temporal action abstraction greatly simplifies the learning problem, it potentially bounds the optimality of the resulting policy as the actions themselves might be non-optimal. A way to increase this optimality, is by parameterizing the actions and using *Policy Search* [166] for optimization. This approach could cope with factors like delay and non-feasible constraints, similar to the work described in [94]. Furthermore, Policy Search can learn a direct mapping between the action parameters and the desired outcome of the action. This approach was described in [165] as *Intra-Option Model Learning* for SMDPs, using the *options* formalism.

Moreover, where actions now are represented by their movements, they can also be learned as a black-box model (i.e., their transition model), which learns how posed constraints in the action directly effects the environment. Actions could then represent a continuous range of outcomes from which the decision maker could choose (i.e., a bounded, continuous action space). This would be similar to, for instance, a human tennis player, that observes the approaching ball and decides where he wants to place it on the court (the action outcome). This is done without explicitly thinking about how he should angle his racket (the posed constraints). With the learned transition models, planning based approaches like Monte Carlo Tree Search could be used as a decision maker. Using Reinforcement Learning to design a decision maker with a continuous action space could

also be considered, although this is highly challenging as finding the optimal action and representing the action-value function will become significantly more computationally demanding.

The transferability of the learned policies and the modeled action primitives to the real table, as discussed in the previous section, appeared highly dependent on the accuracy of the simulator. Although the adaptive nature of the algorithm allows it to deal with certain anomalies, issues like delays, occlusions and small biases prevented the transferability of success rates.

The application on the real set-up is currently limited to policy execution. An actual implementation of continued learning on the real system should be straightforward, apart from some practical problems, such as goal and opponent detection. More importantly, is the fact that generating large data sets on the real set-up is time consuming. If implementations are fully completed however, it could use simulated experience as an initial policy on which it improves during games. It could store several adapted versions, depending on the skill level of its human opponent.

Very distinct *action-primitives* have been used so-far, which only cover a small part of all possible situations. Tackling the full problem could perhaps benefit from defining subgoals, instead of having just the single goal of scoring. For this point of view, hierarchical Reinforcement Learning can provide a framework for analysis, of which an overview can be found in [17].

Chapter 4

Integrating Planning And Execution

The aim of the RoboEarth project is to develop a globally accessible database, that enables service robots to share reusable information relevant to the execution of their daily tasks. Examples of this information are the hierarchical task descriptions, or action recipes*, that represent typical household tasks as symbolic action sequences. By annotating these static action representations with hierarchical planner predicates, they can be interpreted by a* Hierarchical Task Network *planner, such as SHOP2, to compose optimized robot plans flexibly, based on the actual state of the environment and the available capabilities of the robot. To subsequently execute these plans in a household environment, the CRAM executive toolbox is adopted, allowing a tight integration between plan execution and the run-time inference of dynamically updated environment knowledge. The work described here presents the integration of these two planning and execution components into one cohesive framework, tailored for the safe execution of abstract tasks in challenging household environments. The resulting framework is implemented on the AMIGO service robot and a basic experiment is conducted to demonstrate the frameworks integral functionality.*

This chapter is based on *"Integrating Planning And Execution For ROS Enabled Service Robots Using Hierarchical Action Representations"*, R. Janssen, E. van Meijl, D. Di Marco and R. van de Molengraft, International Conference on Advanced Robotics (ICAR),2013

4.1 Introduction

Nowadays robots are mainly used in industry to perform repetitive tasks in predictable environments. Structured manufacturing lines and conditioned workspaces are necessary requirements for robots to safely and accurately perform their instructed tasks. Since robotic systems are getting more socially accepted in our daily lives, they are gradually introduced into more human oriented domains as well, such as the medical and housekeeping sectors, see Fig. 4.1.

Figure 4.1: The AMIGO robot performing tasks in a medical environment.

Bringing robots into human domains is however a challenging task, since these domains are often represented by unpredictable and dynamically changing environments. Furthermore, performing tasks in human-centered environments typically requires robots to have advanced interpretation and reasoning mechanisms, and a well-defined way to predict, or *project*, how their actions change the environment. The reason why humans typically outperform robots on these aspects, is because humans are still far better in exploiting the mechanisms of *memorization* and *communication*: how to perform tasks and their respective outcomes are typically well-known to a human, and if not, they are learned from others or 'googled' on-line.

The RoboEarth project [184] aims to provide robots with such storage and communication mechanisms, enabling the global sharing of information between robots required to efficiently and safely execute their instructed tasks. Examples of information shared through RoboEarth are for instance the maps used for localization and navigation, the object models used for perception and navigation, object ontologies used for reasoning and classification, and task descriptions, or so called *action recipes*, that symbolically describe task related action sequences and constraints. Action recipes currently stored in the RoboEarth database are for instance the task sequences for *serving a drink*, or *setting a table*.

RoboEarth also provides in a common representation for these action recipes through the RoboEarth action language [175]. This language is a formal and grounded representation of the action recipes stored and shared through RoboEarth. One of the main features of this language is the intrinsic embodiment of a semantically annotated taxonomic structure that is used to classify action recipes, allowing class inheritance of recipe properties and straightforward recipe selection based on semantic labels. An example of this can be found in the selection of the action recipe *ServingADrinkWithTwoArms* as a subclass of *ServingADrink*, that describes a more specified action recipe on how to serve a drink when a platform has two arms.

Although the current representation of action recipes enables robots to execute a large variety of pre-programmed tasks, due to their static nature they lack in flexibility when tasks have to be performed in environments with changing constraints, such as when an in-between door is suddenly closed and needs to be opened first before the robot can pass. Another drawback of the current recipe representation is that they do not *project* the state of the environment nor the state of the robot into the future, hence not assuring that an executed action will not interfere with further task completion. A third drawback of the current recipes is that the ordering constraint used to represent the order in which actions are performed, is based on a straightforward numbering technique, impeding the modular (i.e., hierarchical) enclosure of smaller recipes into larger tasks. Finally, since currently the action recipes are not explicitly annotated with action costs, it is not possible to optimize between recipes that accomplish similar tasks.

A solution to the above mentioned drawbacks would be to annotate the current action recipes with planner predicates, such as *pre-conditions*, *effects* and *costs*, and use a hierarchical search algorithm, or *hierarchical task network planner*, to compose optimized plans based on the actual state of the environment and the state of the robot, hereby assuring (based on what was known at plan construction time) that executed actions do not interfere with further plan accomplishment. Composing plans through planning therefore requires a complete overview of the environment at planning time, and adequate re-planning mechanisms in case an insurmountable plan anomaly is detected during plan execution time.

To experimentally validate the above proposed advancements, an integrated system has to be developed that enables the interpretation and use of these planning and re-planning methods, but is at the same time capable of handling small environmental challenges during plan execution. And this is the focus of this chapter; a first design towards the integration of a symbolic, hierarchical planner with a reactive execution engine, that allows the successful accomplishment of abstract tasks by mobile manipulation platforms operating in human domains. The planner has to be capable of interpreting adapted versions of the action recipes as they are defined by the RoboEarth action language, and the execution engine has to be capable of handling dynamic environment properties and small assumption errors; a walking person may have to be tracked, or an object is possibly located at a slightly different position than what was assumed at planning time. In addition, *re-planning* will be incorporated as a recovery mechanism for when plan anomalies cannot be resolved by the execution engine.

The first part of this chapter will give a short summary of the contributions of this work, followed by an overview of related work relevant to these contributions. The succeeding section gives a global overview of the complete system, after which the involved components will be discussed in detail separately. Subsequently, a real-life experiment is presented that, although in a very basic form, describes the systems integral functionality. Finally a discussion is raised, that describes future work related to the improvement of the system and the additions required for the actual reuse of these tasks on a global level through the RoboEarth action recipe database.

4.2 Contributions

The specific contributions of this work are the selection, customization and integration of a hierarchical task network *planner* with a highly reactive and semantically expressive *executive*. Secondly, the system will be equipped with auxiliary components that enable the inference of the *environment state* required for planning and re-planning, and components that enable real-world *human-machine-interaction*. Although the intention of this work is to eventually upgrade to the OWL Semantic Web [22] representation of the RoboEarth action language as earlier described in [175], this work initially focuses solely on the *architecture* required to interpret an already established form of hierarchical action representations and to investigate them for planning and execution in household environments.

4.3 Related Work

In the field of integrated planning and execution architectures for robots operating in dynamic domains there are two well-established systems available; the *LAAS* Architecture [21] based on the *BIP* [18] component design framework and *CLARAty* [139] developed by NASA and the Jet Propulsion Laboratory.

The two-layered *LAAS* Architecture connects its lower level functional layer, implemented in $G^{en}oM$, to the high-level *BIP* component layer [18]. $G^{en}oM$ enables the encapsulation of the robots operational functions in independent modules that manage their execution by asynchronous service requests. The *BIP* methodology describes connections with different priorities between components that, depending on the connection, trigger functions within the $G^{en}oM$ modules. All the *BIP* components run in parallel and can be triggered when needed, allowing the robot to respond immediately to a sudden change in its environment, without having to first initialize certain required behaviors.

The second architecture, *CLARAty*, is also based on a two layered structure. It uses a functional layer for lower level control, which is controlled by a decisional layer that integrates planning, execution and communication. Two different structures for the decisional layer are currently proposed: one describes the combination of a CASPER planner

with a TDL executive [160], and the other is composed of a EUROPA planner combined with a PLEXIL executive [182].

Where the *LAAS* architecture uses its reactive behavior to respond to changes in the environment, *CLARAty* consults software modules to create assumptions that are verified against the perceived state of the world. Both systems therefore establish a certain level of robustness against changes in the environment. They are not however, designed to share task descriptions amongst platforms with different capabilities and have no explicit methods of interpreting nor storing common task descriptions.

In the work of McGann [127] the TREX control framework is adopted to control a Willow Garage PR2 service robot, allowing the robot to handle doors and plugs, while navigating using a topological map. Although the developers briefly mention a desire for the reuse of action *primitives*, there is no further discussion towards the reuse of hierarchical, *composite* robot plans.

Enabling robots with different capabilities to share composite task descriptions requires at first a common representation for these descriptions. As part of the EU funded RoboEarth and RoboHow projects a first implementation is described in [174], where the Semantic Robot Description Language (SRDL) [99] is used to match robot capabilities against the task related components. Although this capability matching enables the filtration of plans that cannot be executed by platforms missing the required capabilities, it does not project the state of the world into the future, therefore not ensuring that a certain robot feature remains available throughout the execution phase of the plan (e.g., a robot-arm might become occupied with holding an object). Its representation also does not allow the calculation of the most optimal plan, in case a task can be executed in multiple ways. Although the language therefore establishes a common and well-defined representation, it is imperative that methods are added for *projection* and *optimization*.

4.4 System Design Motivation

As described in the Introduction, the system should be capable of composing executable plans from hierarchically structured task specifications, and able to execute these plans in a challenging household environment. Based on the literature, a general layout for such a system is represented in Fig. 4.2. The hierarchical action specifications, or in RoboEarth terminology, action recipes, are downloaded from the *Action Recipe Database*. Based on these action recipes, the planner tries to compose a sequence of primitive actions $[O_1...O_n]$ that achieve the instructed task T from world state S_0, where T is received from a *Human-Machine-Interface* and S_0 is received from a *Reasoner*. This reasoner, is capable of inferring the state of the environment and formalizing it in a planner interpretable (symbolic) form. The composed sequence of actions is subsequently exported to the *Executive* and executed, where the Executive handles small assumption errors. The Executive also invokes re-planning, if these errors cannot be overcome on execution level. The next sections discuss the choice and integration details of each of the involved components on a more in-depth level.

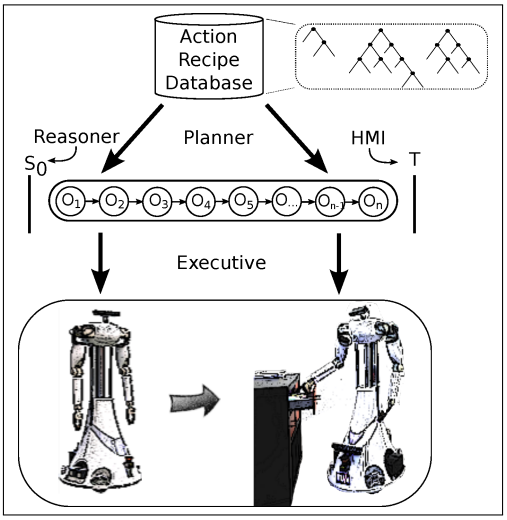


Figure 4.2: The flow through the system with the main components indicated: Action Recipe Database, Planner, Executive, Reasoner and Human Machine Interface.

4.5 Action Recipe Database

The Action Recipe Database contains the action recipes, that symbolically describe the action sequences required to accomplish instructed tasks. The stored action recipes either describe *primitive* actions, i.e., actions that can be considered as grounded robots skills, and *complex* actions, consisting of either other complex and/or primitive actions. Maintaining these action recipes in one central place, such as the globally accessible RoboEarth action recipe database, allows to compare, rank and filter them, and enables reuse across multiple systems in different scenarios. The definition for the action recipes follows from the RoboEarth action language [175], where they are represented by the OWL semantic markup language [141] and modularly arranged in a taxonomic structure, see Fig. 4.3.

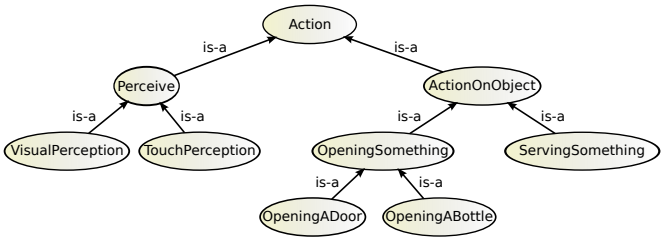


Figure 4.3: Example of the RoboEarth action taxonomy as described in [175].

Although this structure enables the selection of an action recipe based on its taxonomic classification, it does not formally describe under which conditions an action recipe is valid to be used, nor does it represent its decomposition into other complex/primitive actions. Without going into any planning details yet, the work of Nau [137] describes a hierarchical structure in the representation of the planning domain D, hereby enabling the representation of an action recipe as a well known *plan decomposition tree*. An example defined by the 'ServingSomething' action recipe is visualized by the plan decomposition tree in Fig. 4.4.

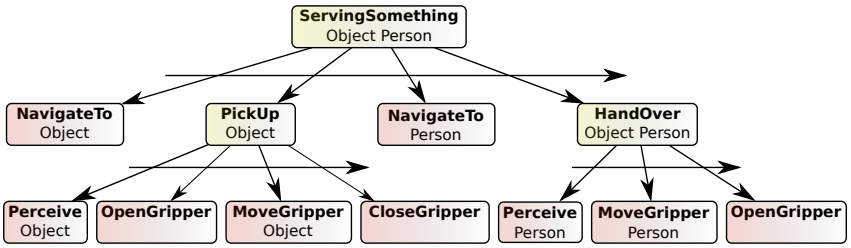


Figure 4.4: A hierarchical plan decomposition tree of the 'ServingSomething' action recipe. Yellow blocks indicate complex actions, and red blocks indicate primitive actions. The left-to-right arrows indicate the order in which the actions are executed.

The RoboEarth action recipes as described in [175] are originally represented in the OWL-DL [141] syntax. To allow a fast first proof of concept of the system, the action recipes discussed in this chapter are represented in the PDDL [126] syntax commonly used for planning. Future work will include the downloading of action recipes from RoboEarth, and the mapping of OWL-DL action recipes onto a planner interpretable syntax. Methods for this mapping are described in the work of Klush [91] and Sirin [161].

4.6 Planner

The goal of the planner is to find a plan P which achieves task T from state S_0 through the planning domain D. A planning algorithm suitable for our system has to meet the following requirements:

1. The planning algorithm must be able to work with a planning domain D that is *hierarchically structured*, because it must be compatible with the RoboEarth language desire of using hierarchical action structures [175].

2. The planner must be fast in solving complex plans, enabling scalability towards the use of a global database and allowing the system to respond quickly if re-planning is requested (which possibly occurs on a high frequency).

3. The planning algorithm should allow for an explicit cost optimization, over plans that achieve similar tasks.

4. The planner must be able to evaluate *external functions calls* during planning, in order to base its assumptions on the most up to date environment state information.

To meet the above requirements, the *Hierarchical Task Network* planner SHOP2 [137] is adopted. First of all this planner is able to solve the planning problem (S_0, T, D) as described above with D being hierarchically structured. Secondly, it is fast in solving complex plans, which is endorsed by the winning of the award for distinguished performance in the 2002 International Planning Competition [114]. Furthermore, SHOP2 has the ability to optimize plans based on a dimensionless cost criterion, and it allows for external function calls in its axioms during planning, see Fig. 4.5.

```
(:- (class-available-in-reasoner ?object)
      (eval (subClass NIL '?object))
      (eval (subClass '?object NIL)))
```

Figure 4.5: SHOP2 axiom: the external function *class-available-in-reasoner*, callable during plan composition.

Finally, SHOP2 is capable of creating *partially ordered* plans, enabling a more efficient plan composition structure (see the example in the next section).

4.6.1 SHOP2 Planning Problem Example

To illustrate how the planner solves the planning problem (S_0, T, D) for creating partially ordered plans, a simple example presents the transporting of two objects, in this case a soda and crackers, from different locations to the same location.

Let the planning problem be described by:

```
T = (transport2 crackers couch_table soda couch_table)

S0 = (avail-arm right)
     (loc-robot entrance_door)
     (loc crackers dinner_table)
     (loc soda kitchen_table)

D = (!loc-robot ?location)
    (!object-in-hand ?side ?object)
    (!object-on-location ?side ?object ?location)
    (place ?object ?location)
    (grasp ?location ?object)
    (navigate ?from ?to)
    (pickup ?object)
    (dropoff ?object ?location)
    (transport ?object ?to)
    (transport2 ?object1 ?to1 ?object2 ?to2)
```

where ‘transport2’ consists of two times the ‘transport’ method. The domain D represents the *operators* (the top three of the list, starting with an exclamation mark) and the *methods* (the remainder of the list). Fig. 4.6 presents the hierarchical decomposition tree of the method ‘transport2’ for the case where only one arm is available.

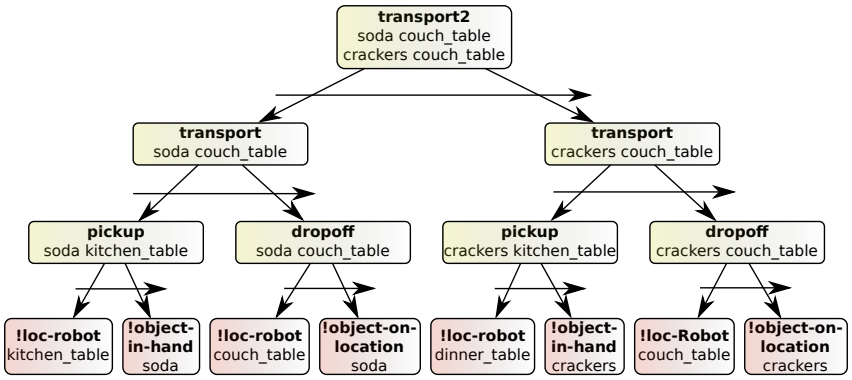


Figure 4.6: The non-interleaved decomposition tree of the method ‘transport2’ when only one arm is available.

Now suppose the robot has two arms available, the right and the left arm. For this example the availability of the left arm will be added as (avail-arm left) to S_0. The resulting plan is given Fig 4.7.

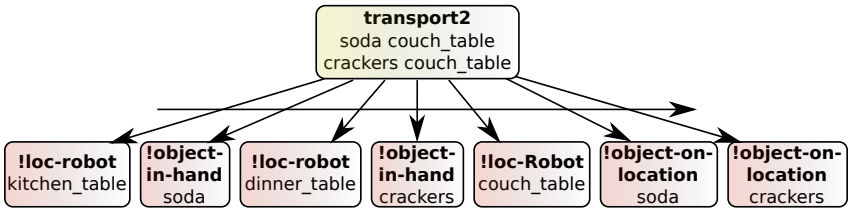


Figure 4.7: This figure presents the operators of the interleaved decomposition tree of the method ‘transport2’ for the case where two arms are available.

The robot will now pickup both objects before returning to the drop off location, demonstrating the benefit of partially ordered task planning. The planner optimizes the number of operators by reducing the accumulated cost, where in this example a unit cost was assigned to each primitive action. In addition, the planner can easily be extended by relating costs to measurable properties of actions, e.g., duration, availability, force, distance, by evaluating external functions calls during planning.

4.7 Executive

The output of the SHOP2 planner is a symbolic sequence of primitive actions. To execute these actions the CRAM [19] toolbox based on Common Lisp is adopted. The reasons for using this toolbox are:

1. It provides tools for the execution of symbolic plans: the domain specific plan language CPL (CRAM Plan Language) and symbolic identities (*designators*) for actions, objects and locations.
2. It is based on a reactive planning approach [34, 58], which explicitly tries to cope with unpredictable and dynamically changing environments. To do this it uses *fluents*, which are system-wide variables that are continuously updated and monitored. Fluents are used in the architecture to trigger for instance re-planning and to start/stop threads when needed.
3. It is based on Common Lisp, by which it can use the standard debugging tools and REPL (*Read-Eval-Print-Loop*) that Common Lisp provides. Because the SHOP2 planner is also implemented in Common Lisp, the domain, state and task definitions can easily be validated through the REPL, resulting in an easy integration of planner and executive.

A complete overview of the system is depicted in Fig. 4.8. The symbolic sequence of action primitives constructed by the SHOP2 planner are mapped onto the domain specific language CPL and executed by the CRAM executive. The CRAM *process modules* convert the symbolic action primitives into parameterized commands that are executable by the robot. The planner *Supervisor* manages the start of the planning process upon receiving a command, collects the information that is required for the planning process, invokes the planning algorithm and instantiates re-planning when this is requested by the executive.

The different components and data structures will be described in more detail in the following paragraphs.

4.7.1 CPL

The CPL language uses semantic control structures to reason about actions through a first-order representation. One of these control structures is the *achieve* function, for which the outcome holds if the process succeeds. Other available control structures are *in-parallel-do* to run different processes in parallel (returns true if all processes succeed), and *try-in-parallel*, which returns true if only one action succeeds.

Fig. 4.9 shows the converted high-level plan for the example task ‘transport2’, used in subsection 4.6.1.

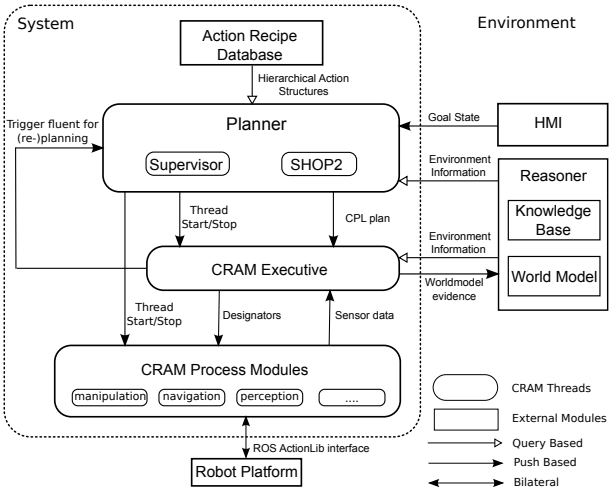


Figure 4.8: A more detailed overview of the integrated architecture. Designators are symbolic representations of objects, locations and actions.

```
(top-level-plan transport2 ()
 (with-designators(
  (crackers ...)
  (soda ...)
  (couch_table ...)
  (dinner_table ...)
  (kitchen_table ...)))
 (achieve '(!loc-robot kitchen_table))
 (achieve '(!object-in-hand right soda))
 (achieve '(!loc-robot dinner_table))
 (achieve '(!object-in-hand left crackers))
 (achieve '(!loc-robot couch_table))
 (achieve '(!object-on-location right crackers couch_table))
 (achieve '(!object-on-location left soda couch_table)))
```

Figure 4.9: The SHOP2 'transport2' plan converted to a CPL top-level plan. The designators are constructed with the *with-designators* command whereas the *achieve* function executes the individual actions.

4.7.2 Designators

Designators are symbolic identities that are bound to real-world concepts. Three different designators are available in CRAM: *object* designators to store information about an object, e.g., type and properties; *location* designators to store the location of an object, which can directly be coupled to the object itself by using for example `(bottle-loc (location(bottle1)))`; and *action* designators which contain information about a task, with the arguments mostly represented by other object and location designators.

4.7.3 Fluents

CPL uses so-called *fluents* to react on changes in the environment. A fluent is a system-wide variable that contains information about sensor data or program events. CPL uses pre-defined functions (e.g., `(whenever fluent body)`, `(wait-for fluent)`) to trigger the agent's reaction on a change of a fluent. In our system, fluents are used for re-planning and to stop/start CRAM threads.

4.7.4 Process modules

Process modules are low level components that convert symbolic designators to parameterized commands that can be sent to the robots control layer. Properties of action designators (*e.g.*, to grasp, to open, etc.) and the object/location designators, e.g., location coordinates, weight etc. are directed to control messages sent to the robots control layer. In our system the robot control layer is implemented by the ROS Actionlib interface [146]. Our demonstrator robot AMIGO currently has 5 different process modules; Manipulation, Navigation, Point-Head, Speech and Perception. The process modules are structured in such a way that similar robots can use similar modules. For example, the manipulation and navigation process modules of AMIGO and the PR2 are almost identical, only the names of the communication channels to the robot control layer are different. For this reason, the process modules can be regarded as the robot's *driver layer*, identical to a driver layer of for instance a Smart Phone or Desktop PC.

4.8 Auxiliary Components

This section describes two auxiliary components, the *Reasoner* and the *Human Machine Interface*, that are required to respectively infer the state of the environment and to deduce a desired task from an abstract user instruction.

4.8.1 Reasoner

The Reasoner provides the system with symbolic information about the state of the environment; it provides the planner with an environment state overview at planning time and it provides the executive with updated information on objects and locations during plan execution. The Reasoner provides a generic *json_prolog* [191] interface to two different sources of information, the *Knowledge Base* and the *World Model*.

The Knowledge Base is a SWI-PL [191] static collection of facts and rules, that are structured in a class ontology describing all common-sense knowledge about the environment. Fig 4.10 gives a small example of this ontology in which one classification and two class properties are depicted.

```
subClass(coke, drinks).
hasProperty(cabinet, object_holder).
classAtLocation(drinks, kitchen).
```

Figure 4.10: Three small Knowledge Base excerpts: an object of class 'coke' belongs to the superclass 'drinks', 'cabinets' can be used as 'object_holders', and anything belonging to the class 'drinks', can typically be found in the 'kitchen'.

The World Model [54] contains a sophisticated tracking and data association algorithm that quantifies streams of sequential measurements, called *evidence*, into unique objects. At its core, the World Model is a multiple-hypotheses filter, able to combine different forms of evidence into a common, dynamically updated world representation. Evidence for the world model can contain spatial information about an object, as well as color, weight, structure, velocity and so on. These attribute-value pairs are associated in the hypotheses tree, and enable object classification based on class attribute information stored in the Knowledge Base.

4.8.2 Human Machine Interface

The Human Machine Interface consists of a standalone version of the Stanford Natural Language parser, which is adapted to map spoken commands onto parameterized PDDL goal tasks. More details about the parser can be found in [84]. Details of the mapping are left out, as they are currently considered to be too pragmatic.

4.9 Basic Experiment

The General Purpose Service Robot challenge of the RoboCup@Home League [180] is chosen as an experimental use-case to demonstrate the basic functionality of our system. The challenge focuses on the following aspects: (1) there is no predefined order of actions to carry out; and (2) environmental reasoning is required to deduce unknown facts. In our example the unknown information consists of the specific locations of objects and locations, such as the 'living_room', 'side_table' etc.

Fig. 4.12 shows the household environment in which the robot has to execute its task. In this example, the task consists of *"Bringing a coke to Erik"*. In the experiment the coke will be located on the 'side_table', but the class position prior stored in the Knowledge Base indicates the coke class to typically reside at the 'cabinet' (*classAtLocation(coke,cabinet)*). During planning the Reasoner is queried for information (availability and location) of the 'coke' and 'Erik'. Their expected locations are returned by the Reasoner and the robot composes and executes the following plan:

When the robot arrives at the cabinet, it tries to perceive the coke for grasping, which is an integral part of the object-in-hand operator. If the object is perceived, it will appear

```
(top-level-plan transport2 ()
    (with-designators(
     (coke ...)
     (couch ...)
     (cabinet ...))
    (achieve '(!loc-robot cabinet))
    (achieve '(!object-in-hand left coke))
    (achieve '(!loc-robot Erik))
    (achieve '(!object-on-location left coke Erik))
```

Figure 4.11: The top-level CPL plan composed for the task of *"Bringing a coke to Erik"*.

in the World Model and the robot can start grasping. If it will not be perceived at the expected location, the action will result in a failure which cannot be handled by the executive[1]. At this stage re-planning is activated by triggering the '*planning-needed*' fluent. A new plan will be composed by SHOP2, based on an updated state of the world. The main difference in this state will be the updated 'coke' prior, for which the expected location has shifted from the 'cabinet' to the 'side-table', based on the false percept at the 'cabinet' (see [68] for more details on falsification).

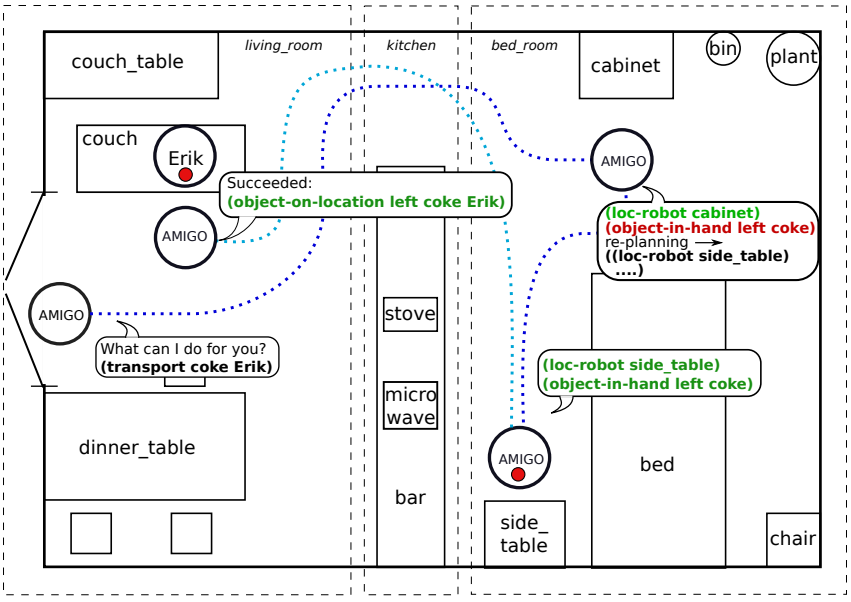


Figure 4.12: Execution of the 'Serving A Drink' task, where the actions in green are achieved and the actions in red have failed.

[1] A full movie of the experiment can be found at http://youtu.be/iFF62gwBaqk

4.10 Discussion and Future Work

This chapter demonstrates an integrated planning and execution framework for domestic service robots, that proposes to extract planning domain knowledge from a common pool of task descriptions. It exploits different techniques to increase robustness, such as re-planning and the real-time referencing of symbolic object and location entities. The first thing that needs to be noted is the limited display of features in the presented experiment; the described use-case does not show the handling of dynamic environment properties, and it also lacks to demonstrate a required change in action sequence after re-planning was invoked by the executive. Furthermore, the cost evaluations made by SHOP2 in this example are based on a unit cost for each action. Future work will integrate function calls that provide in better cost estimates, either by learning [156] or physics simulations [134]. Currently, also the knowledge processing framework KnowRob [173] is being integrated, to enable more extensive reasoning capabilities. On a short term we want to replace the pool of PDDL operators and methods by the globally accessible RoboEarth Action Recipe database, enabling the reuse of actions with other systems through the common OWL-DL action representation. On a longer term, we want to provide this database with feedback on the outcome of actions, allowing filtering and ranking of successful actions.

Chapter 5

Centralized Task Control

This chapter investigates a centralized task controller architecture for robots, allowing time-optimal robot allocation and knowledge reuse at the highest level. To represent robot tasks efficiently and usable for task planning the ontology language for web services OWL-S is adopted as task representation. This language is mapped onto a first-order logic language for action planning in dynamic domains, the Situation Calculus, and implemented as an extension of MIndiGolog, an existing Situation Calculus implementation for multi-robot tasks coded in Prolog. Task planning is achieved through Prolog's first-order theorem proving property, where unbound robot and object parameters are unified with robot and object instances described in environment ontologies. Time optimality is hereby induced, by annotating primitive tasks with durations and selecting the set of Prolog bindings that result in the least time consuming plan. As this work targets the planning of tasks for ROS enabled platforms, a new component model for ROS nodes is proposed together with a grounding ontology designed specifically for the execution of primitive tasks in real-world ROS based architectures. As the architecture is deployed on the RoboEarth Cloud platform, heavy computations can be done here and robots require to run only a small driver layer and a lightweight client interface. A first small-scale experiment is conducted to prove the functionality of system interfaces, task allocation and grounding methods, and to reveal the shortcomings of this work that need to be tackled in future work.

This chapter is based on *"Centralized Task Control With Computational Offloading For ROS Enabled Service Robots Using the RoboEarth Cloud Platform"*, R. Janssen, R. van de Molengraft, H. Bruyninckx and M. Steinbuch, Transactions on Automation Science and Engineering Special Issue on Cloud Robotics and Automation, 2014 (submitted)

5.1 Introduction

In modern day robotics research, cognitive robots are widely investigated as assistive technologies in everyday activities for tasks that are either too boring, strenuous or dangerous to be performed by humans, see Figure 5.1.

(a) (b) (c)

Figure 5.1: Examples of modern day robot assistants. Precise Path Robotics RG3 lawnmower robot (a), Panasonic Hospi delivery bot (b) and Csiro LHD mining robot (c).

Currently, most of these robots are being controlled locally, i.e., each robot receives a private task request, runs its own control program, collects raw sensory data of the nearby environment, processes this sensor data through on-board algorithms, reasons with this processed information on what to do next, and subsequently acts through its actuators. With the dawn of high-bandwidth communication technologies, it is now however possible for robots to communicate sensor information in runtime. It has furthermore become possible, to store and process this vast amount of information through big-data storage facilities and distributed computing platforms. These technologies pave the way for *centralized task control*; one intelligent, knowledge driven system that receives task requests and sensory information on a global level, and controls thousands of connected robots through an optimized multi-robot task planning algorithm. As opposed to robots operating individually, such a system has the following advantages:

- it allows robots to cooperatively perform tasks in an omniscient and optimized manner;
- it allows the knowledge that is used for the planning of robot activities, such as world information, to be collected on a global scale and therefore being most complete and up-to-date. Furthermore, it allows this knowledge to be reused in other task requests,
- it allows fast cross-validation of data, data anomaly detection and data outlier removal;
- it allows a more efficient use of computational effort, as computations can be deployed in a highly optimized computing environment;
- it allows instructed tasks to be performed more robustly, as faulty robots can be replaced directly by similarly capable ones.

The design concept of centralized task control for robots aligns with the concept of *ubiquitous robot networks* [88]. A multi-layer control and sensing architecture that enables interoperability between systems with different hardware and software capabilities. These capabilities can vary from purely virtual, information retrieval services, to real world manipulation activities. The representational language used to enable this interoperability has been established in the work of Juarez [83], where Semantic Web [22] representations are adopted for the unified embodiment of robot topologies. The work of Ha [64] incorporates these web-based representations into the ubiquitous robot network, and adds the Hierarchical Task Network SHOP2 [9] for the automated composition of multi-agent tasks. Their experiment describes task planning for cooperative activities between a mobile robot, a temperature sensor and actuated window blinds. For plan composition, each of these agents was modeled as an abstract web service based on the OWL-S [121] representation of tasks, used to describe document or procedure oriented invocations of services on the Semantic Web.

Although the work of Ha establishes a centralized task planning architecture as described above, they conclude that scalability issues will arise, as in a real world application of their system ad hoc networks with invocable services are expected to be added dynamically. They also conclude that safety and privacy issues may arise as their architecture is fully transparent. Deployment and integration of computationally intense algorithms and task related knowledge bases has in their work not been discussed and most importantly, as their work is closed-source, it cannot be used and advanced upon by the widely established, open-source robotics community.

5.1.1 Contributions

The main contribution of this work is therefore the establishment of a centralized task controller framework for robots, that is scalable and secure, allows deployment of computationally intense algorithms and task related knowledge bases in efficient computing and storage environments, and integrates with existing software design efforts of the open-source robotics community.

5.1.2 Outline

The following section will describe the design of the system, in which the adoption of existing components will be motivated, and in which alterations or added components will be explained. This section will be followed by an experimental use-case, that describes a first demonstration of the system. Conclusions will follow in the end, together with a section on future work required to make the system as general, robust and user friendly as intended.

5.2 System Design

5.2.1 Requirements

The primary goals of the system are 1) to serve as a centralized task controller for a wide variety of robot platforms and computational algorithms, where 2) coordination is dictated by the central controller only. As it is desirable that the involved robot platforms remain lightweight by reducing on-board computing requirements, 3) computational algorithms are supposed to be deployed in a distributed and highly optimized computing environment, instead of running locally on the robots. Furthermore, as the execution of tasks in human domains requires vast amounts of task related knowledge, such as binary models for perception and logic based representations for reasoning, 4) the system is required to have fast and direct access to a knowledge base that is capable of storing and retrieving this information securely and efficiently.

5.2.2 Basic component diagram

A basic component diagram that meets the above requirements is sketched in Figure 5.2

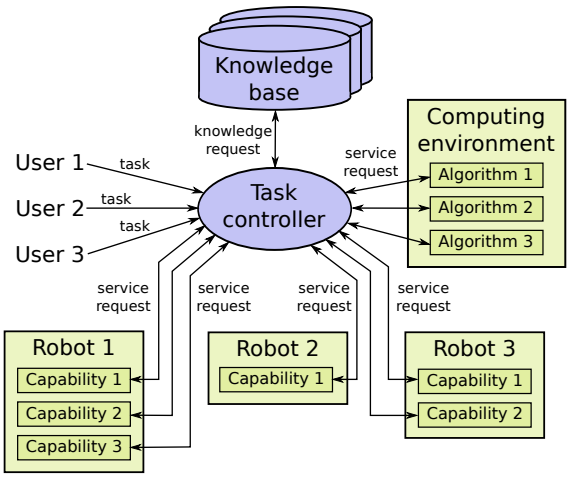


Figure 5.2: Basic component diagram of system requirements.

5.3 Implementations

5.3.1 Communication framework

A key aspect of the component diagram sketched in Figure 5.2, is the communication framework that enables the interfacing between components, and establishes message type protocols. For robotic applications there are several proprietary frameworks available, such as Microsoft Robotics Studio [76] and We-bots [129]. However, as it is required that the supported framework is established as an open-source platform, the well-established robotics middle-ware Robot Operating System (ROS) [146] is targeted.

Amongst several open-source available middle-wares, such as Player [60], Urbi [12] and Orocos [29], ROS can be regarded as currently the most widely used and supported middle-ware. Reasons for its popularity are the vast amount of supported packages and libraries (currently over 3,000), interface support for four commonly used programming languages (C++, Python, Octave and LISP), its peer-to-peer communication approach and its thin messaging layer. This messaging layer currently supports over 400 different message types, such as point-cloud, image, diagnostic and joint-state information.

Attempts have been made to use ROS as a global communication and data storage mechanism in the DAvinCi project [8], where the Apache Hadoop Map/Reduce [47] Framework was used as a data storage and computing environment for the Fast-Slam algorithm [133]. However, the architecture established in the DAvinCi project used only a single computing environment, without any security measures. Furthermore, the DAvinCi project is not publicly available.

A ROS based global communication framework that is publicly available is the RoboEarth Cloud Engine, or *Rapyuta* [72]. Rapyuta is a well-documented and easily installable communication framework, and is developed as part of the effort for global communication and knowledge reuse in the RoboEarth project [184]. As a Platform As A Service (PAAS) framework [39], Rapyuta offers the possibility to:

- deploy one or more secured computing environments, allowing robots to offload their computational algorithms (such as those typically used for grasp planning, mapping and navigation);
- launch multiple processes in parallel (as opposed to Google's App Engine [155]);
- push information from server to robot, through the use of Web-Sockets [185] and serialized Java-Script Object Notation (JSON) string messages;
- communicate messages between multiple ROS masters over the same connection (as opposed to ROS-bridge [43]).

5.3.2 Knowledge base

An added advantage of using Rapyuta is the direct integration with the RoboEarth knowledge repository. This repository contains abstract descriptions of task related information, such as robots, objects, environments and tasks [177]. Furthermore it contains binary data, such as object models and navigation maps. The RoboEarth knowledge repository stores its abstract descriptions in a Sesame database [28], which can be queried through the SeRQL query language [27]. Binary data is stored in the Apache Hadoop File-system [190], allowing efficient and distributed data storage. Both types of knowledge are linked through a relational database, and can be accessed through either a web interface (by humans), or through a REST-full API [152] (by software agents) called `re_comm`[1].

The abstract knowledge stored in the RoboEarth knowledge repository is encoded in the web ontology language OWL [141], or more specifically, in its language variant OWL Description Logics (OWL-DL). OWL-DL provides in maximum expressivity, but remains decidable. This allows the language to be used in most modern day reasoning tools, such as Pellet, Racer, Fact++ or in theorem proving languages, such as Prolog and SQL. A good read on the advantages of using OWL-DL in robotics can be found in the work of Hartanto [67]. For the centralized task planning architecture as proposed in this work, several types of abstract knowledge are required. They will be discussed in Section 5.3.4.

5.3.3 Task controller

The central component in the proposed task planning architecture is the *task controller*. As typically described in robot control literature [4], the goal of the task controller is twofold:

- it needs to identify if incoming user requests can be performed, by attempting to compose a logical course of actions based on the available resources (planning);
- it needs to perform the composed course of actions by interacting with the required resources (execution).

The implementation of these two layers is presented here.

Planning

There are several goal-based methods available for planning, such as STRIPS [57] and planning graphs [26]. However, as these planning methods are proven to be NP-complete

[1] http://wiki.ros.org/re_comm

and very ineffective for large-scale domains, such as those typically found in human environments [31], this work adopts the hierarchical task network planning (HTN) approach as described in the work of Erol [55]. This approach allows a more efficient search in large domains, by providing in plan heuristics as full or partially pre-designed plans.

In RoboEarth, pre-designed plans are called 'action recipes' [120], and identical to HTN methods and operators, RoboEarth action recipes describe *primitive* tasks that are directly executable, and *composite* tasks that are composed of other composite or primitive tasks. In the proposed architecture, a distinction is made between primitive tasks that can be performed on a robot, and primitive tasks that can be performed by one of the computational algorithms.

Planning commences by parsing the action recipes into a *planning language*, that can be interpreted by a planning algorithm. As the action recipes are expressed in OWL-DL, see Section 5.3.4, a planning language and accompanying algorithm need to be selected that are capable of allowing full OWL-DL expressivity. As HTN planning is typically based on PDDL [61] propositional logic, they lack expressivity compared to the description logics semantics of OWL-DL. A language that is capable of expressing description logics semantics is the *Situation Calculus* [124], a high-level, first-order planning language. An accompanying planner implementation, that allows the practical use of this language in planning can be found in *Golog* [103].

As the desire is to plan for tasks in human domains, Golog lacks however in certain required features, such as the ability to plan with concurrent actions, exogenous events or sensed input. For this reason, extensions of Golog have been made, such as ConGolog [45], which allows planning with concurrent actions and exogenous events, and IndiGolog [62], which executes plans iteratively based on sensed input. A recent successor of IndiGolog that allows planning for multi-agent systems, is called MIndiGolog [87]. As this work will focus on the planning of human oriented tasks for multiple robots, MIndiGolog will be used as the foundational planning implementation[2].

As MIndiGolog is a Prolog implementation, planning occurs by the theorem proving property of Prolog (depth-first search). A basic example of a MIndiGolog composite procedure for placing an object at a certain location is given in Listing 1.

```
proc(placeObjAtLoc(Agt,Obj,Loc),
     has(Agt,Obj) // at(Agt,Loc)
     : placeObject(Agt,Obj,Dest)
     : releaseObject(Agt,Obj)).
```

Listing 1: MIndiGolog example of placing an object at a location. The 'has' and 'at' predicates are used as preconditions, and the actions 'PlaceObject' and 'ReleaseObject' are primitive actions. The symbols '//' and ':' indicate control procedures for respectively 'sequential' and 'in-parallel', see [87] for details.

[2] An example of a MIndiGolog domain axiomatization for multiple agents baking a cake, can be found at http://www.rfk.id.au/ramblings/research/thesis/

A valid plan solution for this procedure can be obtained through the following domain axiomatization:

```
agent(amigo_1).
object(sprite_1).
location(table_1).
has(amigo_1, sprite_1).
at(amigo_1, table_1).
```

Listing 2: Domain axiomatization that leads to a plan solution.

Execution of the plan is subsequently performed by the Prolog query `do(placeObjAtLoc(amigo_1,sprite_1, table_1),S0,S)` where 'S0' is the initial state as given in Listing 2 and 'S' is the final state. Predicate 'do' is used in Golog to start the planning process, see [87] for its implementation details.

If in the example domain axiomatization of Listing 2 two agents would have been defined instead of one, e.g., `agent(amigo_1)` ánd `agent(amigo_2)`, two valid plan solutions will be found. A time optimal choice is then made, by accumulating the durations of all involved primitive actions, and choosing the plan solution with the least amount of total time.

Execution

After a viable plan solution is determined by the planning layer, each of the involved primitive actions will be iteratively executed. Execution is performed by a custom made executive module, that has its own knowledge base on action execution (a process called *grounding*), executes the primitive action based on this knowledge, and reports the success of the action back to the Prolog planning module. Depending on the success of the action, the Prolog planner returns the following primitive action to be performed. The executive module is written in Python, as it allows fast development cycles, type introspection and has native bindings for all ROS message types.

Figure 5.3 depicts the integration between planning and execution as an activity diagram, in which the order of steps is as follows:

1. a (typed) task is received at the executive;
2. the executive queries for this task at the planner;
3. the planner queries the knowledge base for available planning knowledge on this task (see Section 5.3.4);
4. the planner tries to find a (time optimal) plan solution;
5. the planner returns the first primitive action of this plan; together with any accompanying bindings;

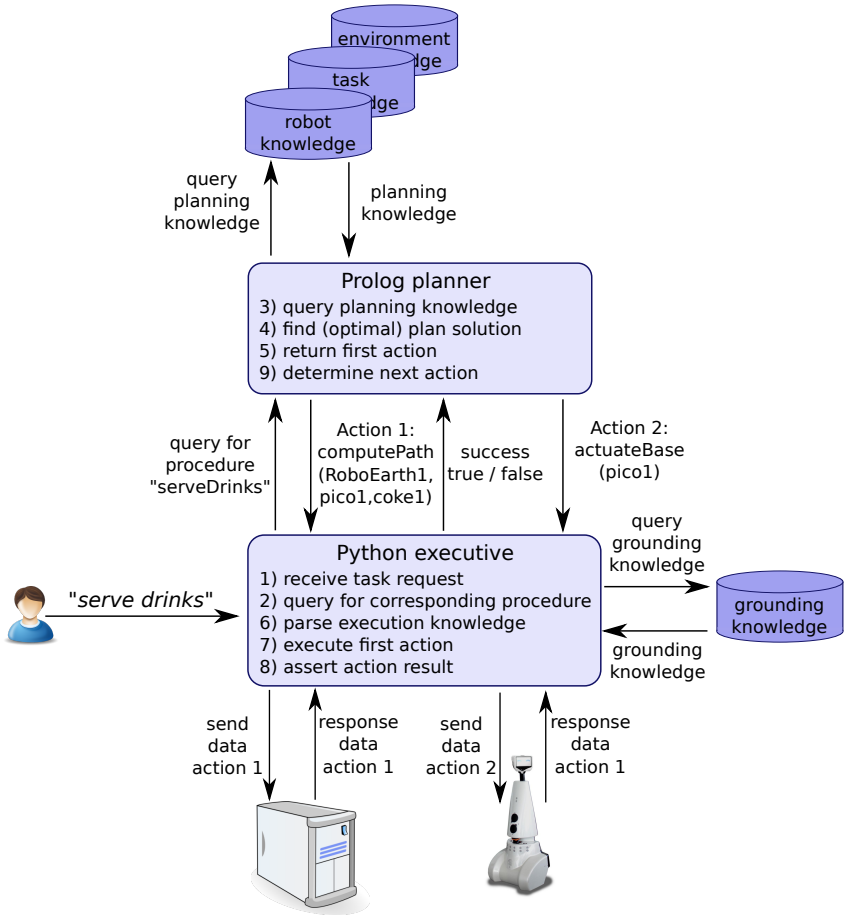


Figure 5.3: The integration between planning and execution.

6. the executive queries for grounding knowledge on this action (see Section 5.3.4);
7. the executive performs the first action, by interacting with the relevant module (either robot or algorithm);
8. the executive asserts the success of the action to the planner;
9. the planner determines the next action ...

5.3.4 Knowledge representations

As the developed architecture intends to plan tasks for diverse algorithms and robot platforms, targeted for operations in human domains, the knowledge that is required for planning and execution consists out of:

- **robot** knowledge, that abstractly describes the capabilities of a robot, such as the ability of using laser, arms or base,
- **environment** knowledge, that describes knowledge generally applicable to the human environment,
- **task** knowledge, that describes abstract representations of primitive and composite tasks,
- **grounding** knowledge, that describes how primitive actions are executed by a robot or algorithm.

Robot knowledge

The *robot knowledge* describes for each connected robot what robot class it belongs to, and what the available capabilities of that class are. These capabilities are either *sensors* or *actuators*, such as a Kinect, laser, arms or base. As an example, Figure 5.4 depicts a topological layout of the TU/e Amigo robot, where each module can be seen as one 'robot capability'.

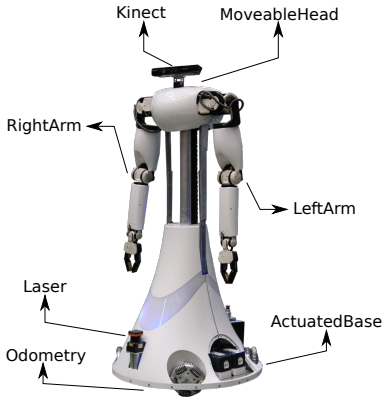


Figure 5.4: Topological layout of Eindhoven University robot class 'Amigo'.

The corresponding robot description for an instance of the 'Amigo' robot class is given in Listing 3.

The robot descriptions are used to match robot capabilities against task required components. This capability matching is performed in the planning language, by adding 'hasSensor' or 'hasActuator' predicates as pre-conditions in the according primitive action. An example, related to the primitive action 'placeObject' from Listing 1, is given in Listing 4.

```
<robot:Amigo rdf:ID="Amigo_1"/>
<owl:Class rdf:about="robot.owl#Amigo">
  <robot:hasSensor
    rdf:resource="robot.owl#Kinect"/>
  <robot:hasSensor
    rdf:resource="robot.owl#Laser"/>
  <robot:hasSensor
    rdf:resource="robot.owl#Odometry"/>
  <robot:hasActuator
    rdf:resource="robot.owl#ActuatedBase"/>
  <robot:hasActuator
    rdf:resource="robot.owl#LeftArm"/>
  <robot:hasActuator
     rdf:resource="robot.owl#RightArm"/>
  <robot:hasActuator
    rdf:resource="robot.owl#MoveableHead"/>
</owl:Class>
```

Listing 3: Robot description for 'Amigo1', instance of robot class 'Amigo'.

```
prim_action(placeObject(Agt,Obj,Dest)) :-
    agent(Agt),
    hasActuator(Agt,rightArm);
    hasActuator(Agt,leftArm).
```

Listing 4: Robot capability matching.

Environment knowledge

General information about the world, such as environment and object properties, is contained in the *environment* knowledge base. Examples are for instance the designated storage, dispose and serve locations for drinks. In Listing 5, an example is given for instance 'Sprite_1' of class 'Sprite' (subclass of 'Drink').

Task knowledge

For tasks, the abstract representation is built upon an existing OWL extension for processes on the Semantic Web, namely OWL-S [121] (formerly named DAML-S). Identical to the RoboEarth action recipes, OWL-S processes can be either one of two things[3]; a *primitive*[4] process, which is directly executable, or a *composite* process, which describes the execution order for other composite or primitive processes. The execution order in composite tasks is dictated by the use of *control procedures*, such as *while-do*, *split-join*,

[3] A third type *simple* process exists, but as this is an abstraction of a composite process it will not be considered here.

[4] Formally called an 'atomic' process in the OWL-S technical description.

```
<environment:Sprite rdf:ID="Sprite_1"/>
<owl:Class rdf:ID="Sprite">
 <rdfs:subClassOf
  rdf:resource="environment.owl#Drink"/>
</owl:Class>
<owl:Class rdf:ID="environment.owl#Drink">
 <environment:hasStorageLocation
  rdf:resource="environment.owl#Refrigerator"/>
 <environment:hasDisposeLocation
  rdf:resource="environment.owl#TrashBin"/>
 <environment:hasServeLocation
  rdf:resource="environment.owl#People"/>
</owl:Class>
```

Listing 5: Abstract knowledge description of a 'Sprite'.

if-then-else, *sequential* and *parallel*[5]. For the evaluation of logical conditions, OWL-S adopts the Semantic Web Rule Language (SWRL) [71], which combines OWL with RuleML, a Semantic Web standard for the evaluation of conditional expressions. As such, SWRL is used within OWL-S for the evaluation of control procedure conditions (such as for if-then-else), and for process preconditions (such as for robot capability matching).

With the Protegé OWL-S modeling tool [53], control procedures can be easily developed by the visual overview and directly imposed logical constraints, see Figure 5.5. Primitive processes are indicated with single surrounding rectangles, and composite processes with double rectangles. On OWL-S modeling level, a distinction is made between primitive processes that have to be executed on a 'robot' platform (if that robot has the proper capability), or if it should be executed by a computational algorithm running on the RoboEarth Cloud Engine (indicated by the 'compute' namespace). Process inputs and outputs, such as 3D point-clouds or joint state information, are not visualized on this level of modeling. These are solely represented in the *grounding* knowledge of each process, see the following section.

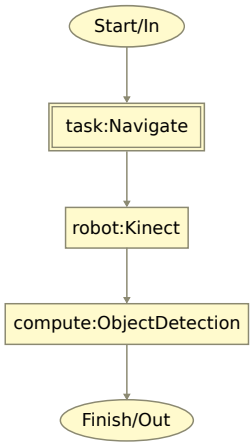


Figure 5.5: Simple OWL-S control procedure for detecting an object, developed in the Protegé OWL-S editor

[5]Extensions to the MIndiGolog domain language have been made in this work, as it natively does not support many OWL-S control constructs, such as *any-order*, *split* and *repeat-while*.

Grounding knowledge

OWL-S enables to model process flows of primitive and composite processes on an abstract level, in a description logics representation. This allows logic based reasoning algorithms, such as planners and schedulers, to use these representations as planner building blocks. These abstract representations however, do not describe how processes are actually executed, called *grounding*. For this, primitive processes require information on implementation, interfacing, parametrization and communication. In the proposed architecture, this information is represented by the *grounding* knowledge, which is composed of an ontology describing process types, messages, parameters and communication channels. The OWL-S ontology by itself provides in a grounding representation suitable for invoking web services through the Web Service Definition Language (WSDL) [37]. WSDL provides in a concrete realization of abstract operations and messages, which can be either document or procedure oriented, and interfaced through either SOAP, HTTP, GET/POST or MIMI. As this work targets the execution of tasks on ROS enabled platforms however, a ROS action/message grounding ontology is specifically developed for this purpose, see Figure 5.6.

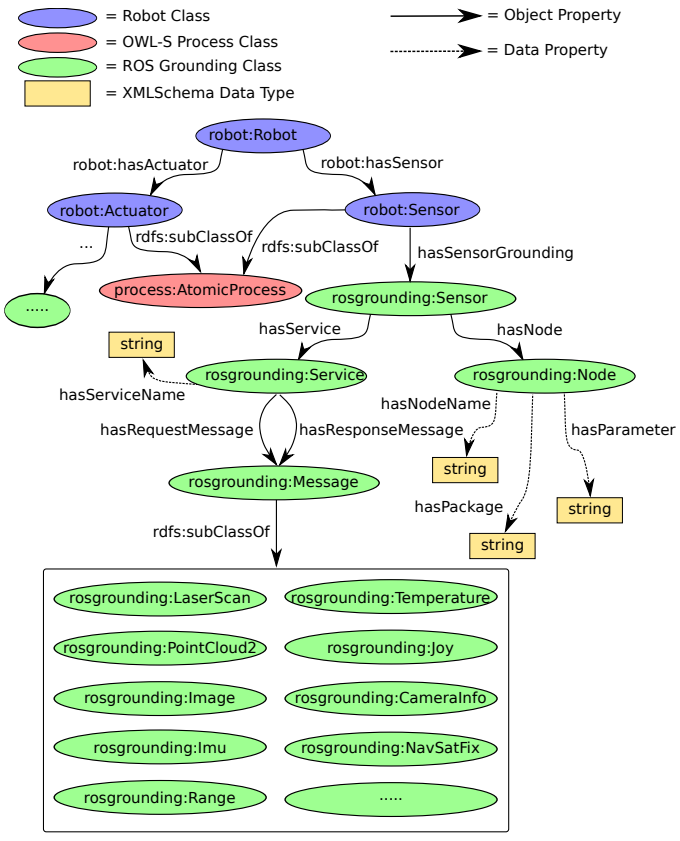


Figure 5.6: Example part of the ROS grounding ontology for a robot sensor.

Grounding ontologies for different middle-wares, such as Urbi or Orocos, are in general also possible to design, but that is beyond the scope of this work. An example OWL snippet of a ROS-grounded task for reading out laser scan messages is given in Listing 6.

```
<owl:Class rdf:about="robot.owl#Laser">
 <rosgrounding:hasSensorGrounding>
  <rosgrounding:Sensor rdf:ID="ReadLaser">
   <rosgrounding:hasService>
    <rosgrounding:Service
     rdf:ID="ReadLaserService">
     <rosgrounding:hasServiceName
      rdf:datatype="XMLSchema#string">/laser
     </rosgrounding:hasServiceName>
     <rosgrounding:hasResponseMessage
      rdf:resource="rosgrounding.owl#LaserScan"/>
    </rosgrounding:Service>
   </rosgrounding:hasService>
  </rosgrounding:Sensor>
 </rosgrounding:hasSensorGrounding>
</owl:Class>
```

Listing 6: ROS grounding snippet for reading out laser scanner messages.

The complete grounding ontology is used for both the grounding of primitive processes on real robots, and for the grounding of computational (primitive) processes running in the computing environment on the RoboEarth Cloud Engine.

5.3.5 ROS component model

Section 5.3.3 describes that all processes are coordinated by the executive. This as opposed to standard ROS architectures, where individual nodes are programmed to communicate individually after certain internal computations have been performed. As this is effective for small dedicated platform architectures, it impedes scalability to multi-robot architectures (considering computational offloading is a requirement). Furthermore, this type of software entanglement is an open invitation to in-code parameterizations that are specific for the application at hand, impeding component reuse.

This work therefore uses an altered ROS node component model, based on Radestock's 'separation of concerns' [147]. This component model allows a clean separation between *coordination*, *configuration*, *computation* and *communication*. The executive in the proposed architecture handles coordination, whereas the ROS framework is used solely for communication. Each node now executes a generic, single computation (or single robot process) which is parametrized based on the configuration parameters found in the *grounding* ontology. As stated in Section 5.3.4 process interfacing is based on the ROS *service* protocol. A time-line sketch of this interfacing is depicted in Figure 5.7.

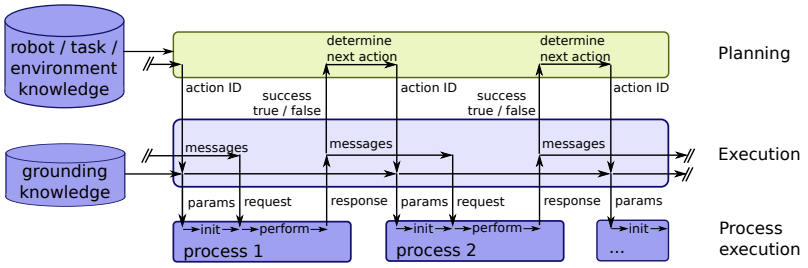


Figure 5.7: ROS component model and process interfacing.

5.3.6 Component deployment

Based on the above mentioned component details, Figure 5.8 depicts a concretized component layout of Figure 5.2.

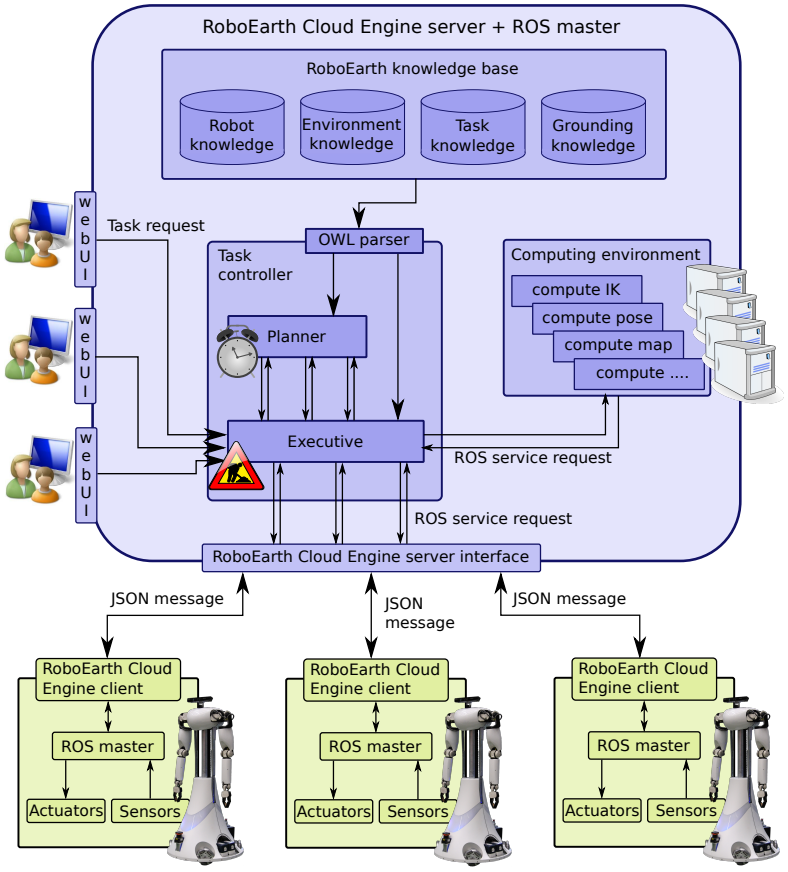


Figure 5.8: Concretized component layout of Figure 5.2.

5.4 Experimental use-case

To validate the functionality of the proposed system, an experimental use case has been devised. The experiment describes two robots, the Eindhoven University Amigo and Pico, see Figure 5.9, serving and cleaning up drinks at a 'cocktailparty' in the Eindhoven University robotics lab.

Figure 5.9: TUE/e Amigo (left) and Pico (right).

5.4.1 Experiment description

The OWL-S top-level task for the 'cocktailparty' task[6], is depicted in Figure 5.10.

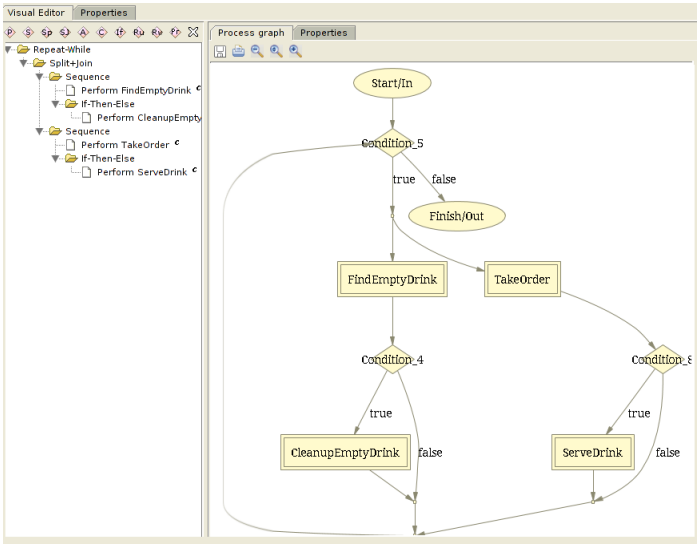


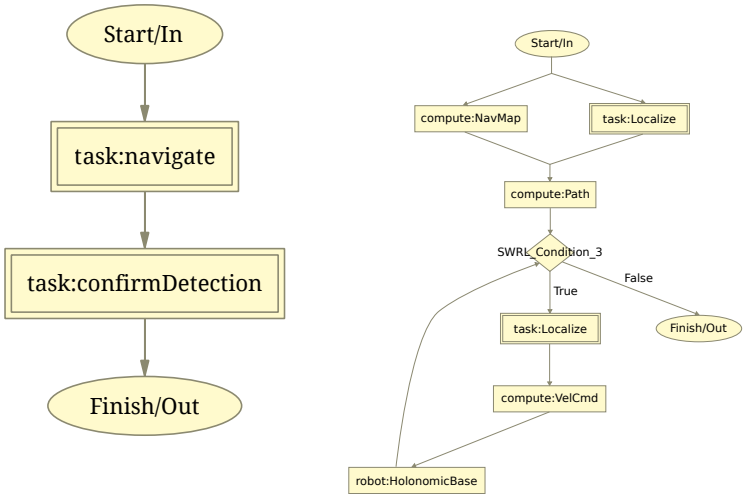
Figure 5.10: Top level control flow for 'cocktailparty' task.

[6]Designed with the Protegé OWL-S editor plug-in

The top-level cocktailparty task consists out of 4 main subtasks;

- 1) 'TakeOrder'
  - 2) if an order is received, 'ServeDrink'
- 3) 'FindEmptyDrink'
  - 4) if an empty drink is found, 'CleanupDrink'

The experimental description here will focus on one subtask, namely the 'FindEmptyDrink', see Figure 5.11.



(a) (b)

Figure 5.11: Control flow for 'FindEmptyDrink'(a) and subtask 'Navigate'(b).

For this experiment, the described robot capabilities for Amigo and Pico are identical:

- **hasSensor**(amigo_1,Kinect)
- **hasSensor**(pico_1,Kinect)
- **hasSensor**(amigo_1,Laser)
- **hasSensor**(pico_1,Laser)
- **hasActuator**(amigo_1,Base)
- **hasActuator**(pico_1,Base)

The accompanying services running on the robots are depicted in Table 5.1.

service	inputs	parameters	outputs
Kinect	none	none	senMsgs/Image senMsgs/CamInfo senMsgs/Image bool success
Laser	none	none	senMsgs/LaserScan bool success
Base	geoMsgs/Twist	none	bool success

Table 5.1: Services running on the robots.

Furthermore, six computational nodes have been deployed in the Roboearth cloud engine, for which their inputs, outputs and parameterizations are listed in Table 5.2:

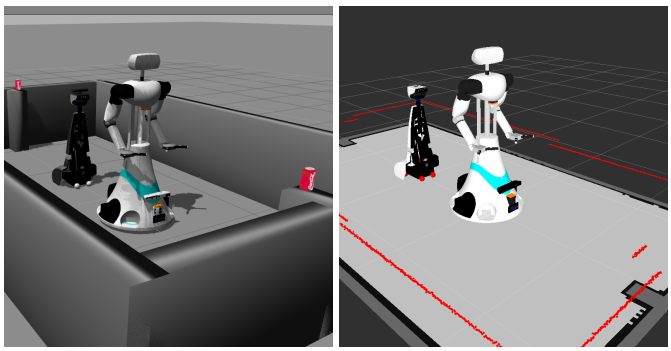
- **LocMap:** computes a nav_msgs/OccupancyGrid used for localization,
- **NavMap:** computes a nav_msgs/OccupancyGrid used for navigation,
- **Path:** computes a nav_msgs/Path from location parameters A and B. A and B are bound by the planner to initial robot and drink locations obtained from the 'environment.owl' knowledge base,
- **Pose:** computes a geometry_msgs/PoseStamped that indicates the current position of the robot,
- **Detection:** detects an object, parameterized by its HUE values [151]. Successful detection is concluded from the action return value for 'success',
- **VelCmd:** computes velocity commands, based on desired path and current pose. Returns true only if the final point in the path is reached.

service	inputs	parameters	outputs
LocMap	none	locMap.yaml	navMsgs/OccGrid bool success
NavMap	none	navMap.yaml	navMsgs/OccGrid bool success
Path	navMsgs/OccGrid	robotPose(x,y,θ) targetPose(x,y,θ)	navMsgs/Path bool success
Pose	navMsgs/OccGrid senMsgs/LaserSc. geoMsgs/PoseSt.	robotFrame	geoMsgs/PoseSt. bool success
Detection	senMsgs/PointCl. senMsgs/CamInfo	objectHUE.yaml	bool success
VelCmd	navMsgs/Path geoMsgs/PoseSt.	maxVelLinear maxVelAngular	geoMsgs/Twist bool success

Table 5.2: Computational nodes launched in RoboEarth Cloud Engine.

5.4.2 Simulator

To allow a fast development cycle and easy parameter tuning (such as the parameters for maximum robot velocities and object HUE values), a simple test environment has been devised in the ROS Gazebo simulator, see Figure 5.12.



(a) (b)

Figure 5.12: Gazebo simulator (a) and Rviz visualizer (b).

In this test environment, the two robots are spawned together with two 'empty' drinks. The goal is to execute the 'FindEmptyDrink' task as devised in OWL-S, where both robots are supposed to search for the empty drinks at the locations that nearest to their current positions.

5.4.3 Real world

In the real-world version of the experiment, RoboEarth client interfaces are deployed on the robots, allowing the experiment to be conducted in a real lab environment, see Figure 5.13.

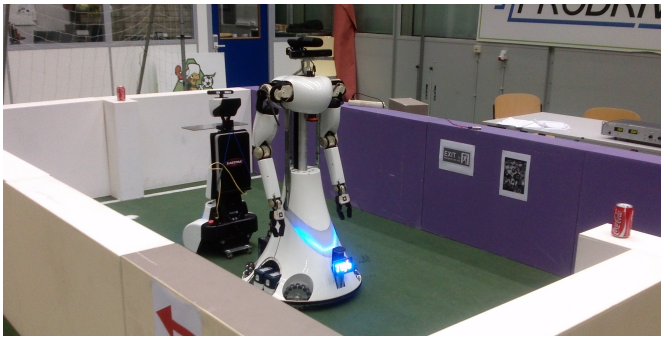


Figure 5.13: Real world experiment initial positions.

Figure 5.14 shows the two robots reaching their final positions, where the 'empty' drinks are positively detected[7].

(a) (b)

Figure 5.14: Real world experiment final positions.

Planner data has been logged, and shows the following incremental plan execution:

```
do [NavMap(amigo_1,"tue_lab")] at time 19.02
do [NavMap(pico_1,"tue_lab")] at time 19.02
do [LocMap(amigo_1,"tue_lab")] at time 20.41
do [LocMap(pico_1,"tue_lab")] at time 20.43
do [Laser(amigo_1)] at time 21.98
do [Laser(pico_1)] at time 22.04
do [Pose(amigo_1)] at time 22.19
do [Pose(pico_1)] at time 22.25
do [Path(amigo_1,coke_1)] at time 23.98
do [Path(pico_1,coke_2)] at time 24.03
do [LocMap(amigo_1,"tue_lab")] at time 25.43
do [LocMap(pico_1,"tue_lab")] at time 25.55
do [Laser(amigo_1)] at time 26.9
do [Laser(pico_1)] at time 26.98
do [Pose(amigo_1)] at time 27.05
do [Pose(pico_1)] at time 27.17
do [VelCmd(amigo_1)] at time 31.86
do [VelCmd(pico_1)] at time 31.96
do [Base(amigo_1)] at time 33.29
do [Base(pico_1)] at time 33.52
do [LocMap(amigo_1,"tue_lab")] at time 34.89
do [LocMap(pico_1,"tue_lab")] at time 34.94
....
do [Kinect(amigo_1)] at time 57.59
do [Detection(coke_1)] at time 58.71
do [Kinect(pico_1)] at time 61.04
do [Detection(coke_2)] at time 62.45
```

Listing 7: Planner log.

[7] A video of the experiment can be found at http://youtu.be/4jCGcRs6GZI

What can be concluded from this log, is that the time between two consecutive localization steps (started when the second 'LocMap(Agt,Env)' is performed), is approximately 9.46 [s], and hence, has an update frequency of ∼0.1 Hz. This is a lot lower than native ROS localization components, such as AMCL[8], which typically run at 20∼40 Hz. This is caused primarily by the service based interface, which can be considered much slower than AMCL's topic based interface.

Furthermore the packet size per service on 'amigo_1' has been logged, see Table 5.3[9].

service	**sent**	**received**
Laser	4136	0
Base	1	48
Kinect	2150929	0

Table 5.3: Communication data on client 'amigo_1' (in bytes).

Combining the planner log from Listing 7 with the packet sizes displayed in Table 5.3, results in an average data transfer rate of 442 Bps (Bytes/second) for one combined localization and navigation step. This is however based on the earlier concluded update rate of 0.1 Hz. If the update frequency of the service interface can be improved, and communication updates can be scaled up to a rate of 30 Hz (comparable to that of AMCL), an average data transfer rate of 130 KBps (KiloBytes/second) will be obtained.

For dynamic look-and-move visual servoing applications [73], that typically require point-cloud and image update rates of ∼30 Hz [38], data transferring requires a significantly larger amount of communication bandwidth. If the Kinect service is called at 30 Hz, this will result in a data transfer rate of 61.5 MBps (MegaBytes/second). For current wireless router protocols, such as wireless B,G and N, these speeds can not be achieved, as their maximum data transfer rates are 1.4, 6.8 and 31 MBps respectively. This means that for visual servoing purposes, the proposed interfacing methods are not suitable.

Computational efforts have also been logged on both client robots, and CPU usage does not exceed 4% during navigation (both are Intel I5 Quad-Core processors). Only when the Kinect service is called upon, CPU usage increases temporarily to 170%, distributed over 2 cores.

In future work, this demo will be expanded to the full 'cocktailparty' demo, including user interaction for taking orders, and manipulation, for the actual serving of drinks. Also the environment will be upscaled to the full TU/e robotics lab, increasing the work space and the number of available order, serve, storage and dispose locations.

[8] http://wiki.ros.org/amcl

[9] As composed plans and hardware components are identical to both robots, the communication data for 'amigo_1' is assumed to be identical to the communication data of 'pico_1'

5.5 Conclusions & Future Work

The work described in this article presents a centralized control architecture, that can be used for the deployment of ROS enabled robots operating in human domains. A first experiment is conducted that, although in a very first basic form, indicates the functional success of this first implementation and the usability of such an architecture. For future work, there are however a few enhancements that can be made to the current implementations and design choices.

A first point of remark should be pointed towards the OWL-S editor plug-in for Protegé, that was used to model the required control procedures. The plug-in shows quite some stability issues, resulting in frequent Protegé crashes and unresolvable modeling anomalies. This translates to a low user friendliness of the editor, and therefore needs to be solved by inspecting and debugging the plug-in. A secondary desire here, could be to upgrade the plug-in to be used in the latest Protegé version (4.3, currently) as this version, as opposed to version 3.5 used for the plug-in, supports easier modeling of classes, object properties and cardinality restrictions.

What can be further noticed, is that the current interface to the robot components and the computational algorithms running on the RoboEarth Cloud Engine using ROS services, is not as fast as required for certain procedures. As can be seen in the OWL-S process flow for 'Navigate' (Figure 5.11b), is that with every cycle the subtask 'Localize' is called upon, which computes the robots pose by receiving its laser-scanner data. For standard ROS navigation architectures this laser-scanner data is processed at 30 Hz, whereas with the service call implementation only a rate of approximately 0.1 Hz can be achieved. This results in having to stick with very slow movements of the robot ($<$1 cm/s for translational, and $<$0.01 rad/s for angular speeds), as otherwise its recursive location estimation will get lost. As concluded in Section 5.4.1, these low update rates combined with high bandwidth requirements for the Kinect data, also make this system impractical for dynamic look-and-move visual servoing applications.

In addition to this comes the fact that with this first implementation, no error recovery behaviors are implemented. This means that when a certain action fails, there are currently no recovery behaviors that can bring the robot in a previous (safe) state from which it can retry. Investigations have to be done to design such error recovery mechanisms, starting of with first being able to monitor 'what went wrong'. These mechanisms are of course also to be represented in OWL-S, making them reusable and shareable amongst other platforms with identical topologies.

As for the robot capability matching a first prototype was included, that allows a straightforward modeling of required components in OWL-S. Matching these against the capabilities found on a robot, was achieved by manually design of the 'robot' knowledge ontology. These robot descriptions can however also be derived from robot configuration files already deployed on the robot. Work in this direction has been done by Kunze, in the Semantic Robot Description Language (SRDL) [99].

A final point that needs to be addressed, is the lack of dynamic state representations in the current architecture. Although the robot position was dynamically updated in the task planning component as an internal state variable, currently there is no advanced mechanism available to track objects over time, and to associate incoming measurements with previously identified objects. For this world model representation, object tracking and association algorithms, such as the one described in [54], should be promising additions.

Chapter 6

Conclusions and recommendations

6.1 Conclusions

This thesis presents an architecture suitable for the control of multiple robots operating in the human domain, where required key methods for segmentation, tracking, learning and planning have been investigated that allow robots to successfully advance into this domain. This thesis therefore extends into several robotic research areas, discussing a variety of methods useful for the centralized task control of cognitive robots operating in human domains.

On the use of the IMM tracking method discussed in Chapter 2, the conclusion was drawn that it outperforms a Kalman filter for maneuvering targets if the maneuvering index (the ratio between process noise and measurement noise times the squared sample time) exceeds 0.5. When applying this IMM tracking method in a globally updated tracking architecture, such as proposed in Figure 1.6, it will therefore be expected to show significant performance improvements over a Kalman filter compared to a small scale tracking application, such as presented in Chapter 2. This can be motivated by the fact that the ratio between process noise and measurement noise will remain the same (as the sensor observing the target and the accuracy of the target model remain identical), but that the sample time for global applications will drastically increase because of global communication delays and processing times, compared to communication delay and processing times on an embedded application. For this reason the maneuvering index will increase, favoring the IMM over Kalman filtering for global tracking applications.

The application of learning methods in Chapter 3 concluded that learning outcomes are

highly sensitive to the design of the reward function, and that for its design human heuristics are involved that need to be accurately applied. The use of function approximation significantly increased the performance of learned policies compared to grid-based policies, and decreased learning convergence times. These times were further decreased by improving sample efficiency through the use of eligibility traces, especially for learning trials performed in noisy environments. Furthermore, it was proven in these noisy environments that learned policies outperform hand coded policies, especially when also unforeseen environment changes occur. Finally, this chapter also concluded that transferability issues arise and policy performance degrades, if simulation environments developed for learning do not match with the real-world applications on which the learned policies will be applied. One important mismatch that can occur for vision based learning applications is *camera delay*.

For the high-level, multi-robot experiments conducted in Chapters 4 and 6, a more general point to be addressed is the amount of integration work required for the involved experiments. It has become clear that the level of a theory relates to the amount of work necessary to translate concerning concepts into full demonstrative applications. For the segmentation and tracking methods discussed in Chapter 2, mostly low level perception and actuation components were required, such as image grabbing, segmentation, tracking and motion control, and verification of the implementations and quantitative results were obtained through a relatively confined and isolated experiment. Subsequent validation for the Greedy-GQ(λ) learning algorithm as proposed in Chapter 3 involved additional steps, such as the design of a simulator mimicking the real world setup, performance measure design, policy learning and policy execution. Because of these added steps, relevant results to evaluate the theory at hand and to demonstrate its functionality on a real world platform required more integration work and became significantly more difficult to obtain. The planning algorithm that was investigated in Chapter 4 required even more components, and thus, more implementations were required to demonstrate the applicability of this integrated system. Furthering this work into the multi-robot, cloud based task scheduling architecture that was presented in Chapter 5 required even more integration work, requiring a total of at least 48 man-months to be accomplished.

As with any design that involves a variety of software components, it can be observed that the extensive number of involved components and their algorithmic complexity are hard pledges for software reuse and open source software licensing[1]. Although Chapter 5 critiques their currently used component model, the ROS middle-ware in general has proven to be an excellent initiative in this direction. Proof of this can be found in the more than 450 software packages included in their current release (Groovy), and the vast amount of accompanying software documentation. The company that initiated the ROS platform (Willow Garage) has performed well in infrastructural maintenance and support, and by now has established a firm base in both academic and industrial robotics research.

The main advantage of having a common communication interface between different software components is that of the ‘software language problem’. ROS provides in such

[1] http://opensource.org/licenses

interfaces, by forcing its developers to develop interface wrappers for software languages commonly used in robotics, such as Python, C++, Lisp, Java and Lua. Furthermore, representational and reasoning languages, such as OWL and Prolog are supported within ROS through the integration of third party libraries, such as KnowRob [173] and ROSprolog[2].

In practicality, this thesis has shown the first applications of several mechanisms useful for the control of cognitive robots operating in human domains, i.e., domains that are unstructured and unpredictable (although unpredictability can be seen as a form of unstructuredness in the time domain) and characterized by object models and task concepts related to human domains. Although the demonstrated systems can be conceptually applied to real world human domains, they do require performance improvements and added mechanisms for fault recovery to make them as robust and stable as required by these domains. Similar to the introduction of desktop computers and mobile phones, robot platforms and the software that controls them needs to have a high level of robustness and user friendliness. Considering the fact that these systems will not just 'think' (like computers), but will also 'manipulate', it is even more important for them to derive the right interpretations and to make the right decisions. For this reason, it is therefore difficult to predict when the first full autonomous systems will be deployed in our everyday lives. Fortunately, as with phones and desktop computers, they do not necessarily have to be as complex as what researchers at this point in time already desire them to be. Examples in this are the simpler robot types that are currently mowing our lawns and vacuum cleaning our living rooms. For more advanced activities, tele-operation, such as currently being applied to underwater welding and bomb disposal robots, is currently a good transitional step to full autonomy.

6.2 Recommendations

Chapter 2 of this thesis demonstrated an interesting alternative to established methods that are currently mostly used in tracking applications. The focus of this chapter was to compare distinct methods used specifically for the purpose of target tracking of a highly agile, maneuvering target. To integrate the proposed Interacting Multiple Model filter however in the cloud based task scheduling architecture as proposed in Chapter 1 however, several points need to be addressed in addition. As the proposed IMM filter in Chapter 2 only deals with tracking, a first point to be addressed is the current lack for adequate *data association* techniques, that associate received measurements with the appropriate existing object instances. A recommendation in this context is therefore the inclusion of methods for data associating, where the work of Bar-Shalom [15] can be seen as a cornerstone in this area, especially because it combines tracking with data association techniques. A more extensive, state of the art overview of tracking and data association techniques can be found in the work of De Laet [46]. To furthermore apply this type of tracking and association techniques to the human domain specifically, a second point that needs to be addressed is the required *anchoring* techniques that need to be integrated.

[2] http://wiki.ros.org/rosprolog

Anchoring is a technique in which multiple object attributes, such as position but also color, weight, etc. are linked against incoming measurements. An introductory overview, and extended work in this direction can be found in the work of Saffiotti [42]. The work of Elfring [54] combines these methods, and provides in a full ROS compatible software implementation called *Wire*[3].

Chapter 3 discussed a learning method that allows systems to improve upon their decision making capabilities. Because the focus in this chapter is on the application and evaluation of the Reinforcement Learning method Greedy-GQ(λ) on an application that has a smaller state and action space than the proposed architecture from Figure 1.6, extensive effort needs to be made to scale the learning problem up to a size similar to that of a full global learning architecture. When applying Greedy-GQ(λ) on an application of such a global size, the question would be if temporal difference learning methods are still viable to be performed with respect to scalability and convergence time constraints. For such a large scale application it will probably be more worthwhile to investigate learning methods for data classification and data outlier removal, for which *data mining* techniques such as those addressed in [32] can be used, as these enable the analysis of large data sets for unknown data patterns (data clustering) and the detection of unusual patterns (anomaly detection). An extensive overview of data mining techniques from a database perspective is given in [65]. Furthermore this chapter concludes that the design of an accurate reward function based on human heuristics is a difficult task, which can be assumed even more challenging for a large application as proposed in Figure 1.6. As a solution, meta-learning of the reward function can be made an automated process through the use of evolutionary algorithms, such as described in the work of Sumino [164].

The work described in Chapters 4 and 5 focused on planning and execution for robots operating in human domains, where the latter extends to the integration of semantic web task representations. Here, a reference was made to the *ubiquitous robot network* as described in [88], where robot platforms are modeled as modular platforms with abstract component representations. As opposed to the WSDL based grounding implementations used there, this thesis focuses on the grounding of robot tasks on ROS [146] middle-ware, as it currently can be regarded as one of the most used and supported middle-wares for robots. However, to allow true integration with existing web services, home automation systems and database information retrieval systems that are not running on ROS, it will be necessary to provide additional grounding definitions that align with the invocation methods of these systems. A good example can be found in the work of Ha [64], where WSDL is used as a common grounding method for the invocation of several robotics and home automation systems. Unfortunately however, their work is not openly accessible.

A concern that has not yet been addressed in this thesis are the ethical and privacy related concerns that inevitably will need to be addressed in a near future. As currently large discussions are going on about the capturing and storage of data coming from the public domain, the development of cloud based robot control architectures needs to take place in these discussions in an early stage, as their functional existence depends on the availability of this data, and on the infrastructural investments that are currently being

[3] http://wiki.ros.org/wire

made to support and process this data. No definite answers to these questions currently exist, but it is clear that society demands for them and that scientific developments cannot continue without having established the ethically right set of privacy related protocols. Fortunately, as with any world-altering technological development, through rational discussion and creative (re-)engineering, consensus will *always* be found.

Bibliography

[1] *World Population Ageing, 1950-2050.* United Nations Publications, Room DC2-853, 2 UN Plaza, New York, NY 10017, 2002.

[2] Rainer Bischoff 0004, Tim Guhl, Erwin Prassler, Walter Nowak, Gerhard K. Kraetzschmar, Herman Bruyninckx, Peter Soetens, Martin Hgele, Andreas Pott, Peter Breedveld, Jan Broenink, Davide Brugali, and Nicola Tomatis. Brics - best practice in robotics. In *ISR/ROBOTIK*, pages 1–8. VDE Verlag, 2010. ISBN 978-3-8007-3273-9.

[3] Russell J. Abbott. Knowledge abstraction. *Commun. ACM*, 30(8):664–671, August 1987. ISSN 0001-0782. doi: 10.1145/27651.27652.

[4] Rachid Alami, Raja Chatila, Sara Fleury, Malik Ghallab, and Félix Ingrand. An architecture for autonomy. *I. J. Robotic Res.*, 17(4):315–337, 1998.

[5] Allied Vision Technologies. Prosilica drivers, August 2013.

[6] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1.* IBM, 2003.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010. ISSN 0001-0782. doi: 10.1145/1721654.1721672.

[8] Rajesh Arumugam, Vikas Reddy Enti, Bingbing Liu, Xiaojun Wu, Krishnamoorthy Baskaran, Foo Kong Foong, Appadorai Senthil Kumar, Dee Meng Kang, and Wai Kit Goh. Davinci: A cloud computing framework for service robots. In *ICRA*, pages 3084–3089. IEEE.

[9] Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dana S. Nau, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *CoRR*, abs/1106.4869, 2011.

[10] Peter Auer and M. Long. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:2002, 2002.

[11] F. Augugliaro, A. Schoellig, and R. D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1917–1922, 2012.

[12] Jean-Christophe Baillie. Urbi: towards a universal robotic body interface. In *Humanoids*, pages 33–51. IEEE. ISBN 0-7803-8863-1.

[13] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995.

[14] Prith Banerjee, Rich Friedrich, Cullen Bash, P. Goldsack, Bernardo A. Huberman, J. Manley, Chandrakant D. Patel, Parthasarathy Ranganathan, and A. Veitch. Everything as a service: Powering the new information economy. *IEEE Computer*, 44(3):36–43, 2011.

[15] Y. Bar-Shalom and T.E. Fortmann. *Tracking and data association*, volume 179 of *Mathematics in Science and Engineering*. Academic Press Professional, Inc., San Diego, CA, USA, 1987. ISBN 0-120-79760-7.

[16] Y. Bar-Shalom, S. Challa, and H.A.P. Blom. Imm estimator versus optimal estimator for hybrid systems. *Aerospace and Electronic Systems, IEEE Transactions on*, 41(3):986 – 991, july 2005. ISSN 0018-9251.

[17] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[18] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *Fourth IEEE International Conference on Software Engineering and Formal Methods, SEFM*, pages 3–12, 2006.

[19] M. Beetz, L. Mösenlechner, and M. Tenorth. CRAM – A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1012–1017. IEEE, 2010.

[20] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[21] S. Bensalem, M. Gallien, F. Ingrand, I. Kahloul, and N. Thanh-Hung. Designing autonomous robots. *Robotics & Automation Magazine, IEEE*, 16(1):67–77, 2009.

[22] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[23] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1st edition, 1996. ISBN 1886529108.

[24] J.A. Besada, J. Garcia, G. De Miguel, A. Berlanga, J.M. Molina, and J.R. Casar. Design of imm filter for radar tracking using evolution strategies. *Aerospace and Electronic Systems, IEEE Transactions on*, 41(3):1109 – 1122, july 2005. ISSN 0018-9251.

[25] H.A.P. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with markovian switching coefficients. *Automatic Control, IEEE Transactions on*, 33(8):780 –783, aug 1988.

[26] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1–2):281–300, 1997.

[27] Jeen Broekstra and Arjohn Kampman. Serql: An rdf query and transformation language. August .

[28] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In Ian Horrocks and James Hendler, editors, *Proceedings of the first Int'l Semantic Web Conference (ISWC 2002)*, Lecture Notes in Computer Science, pages 54–68, Sardinia, Italy, May . Springer Verlag. ISBN 978-3-540-43760-4. doi: 10.1007/3-540-48005-6_7.

[29] Herman Bruyninckx. Open robot control software: the orocos project. In *ICRA*, pages 2523–2528. IEEE. ISBN 0-7803-6578-X.

[30] Thomas Brunl. *Embedded robotics - mobile robot design and applications with embedded systems (3. ed.)*. Springer, 2008. ISBN 978-3-540-70533-8.

[31] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69:165–204, 1994.

[32] Peter Cabena, Pablo Hadjinian, Rolf Stadler, Jaap Verhees, and Alessandro Zanasi. *Discovering Data Mining: From Concept to Implementation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998. ISBN 0-13-743980-6.

[33] Arthur Carvalho and Renato Oliveira. Reinforcement learning for the soccer dribbling task. *CoRR*, abs/1305.6568, 2013.

[34] D. Chapman. Planning for conjunctive goals. *Artificial intelligence*, 32(3):333–377, 1987.

[35] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 216–217, 2008.

[36] L. Chen et al. Robotic foosball table, 2007.

[37] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. World Wide Web Consortium, Recommendation REC-wsdl20-20070626, June 2007.

[38] Hyungsuck Cho. *Opto-Mechatronic systems handbook: techniques and applications*. CRC Press, Abingdon, 2003.

[39] Beth Cohen. Paas: New opportunities for cloud application development. *IEEE Computer*, (9):97–100.

[40] Community Project. Open CV, August 2013.

[41] S. Connelly et al. The autonomous foosball table, September 2007.

[42] S. Coradeschi and A. Saffiotti. An introduction to the anchoring problem. *Robotics and Autonomous Systems*, 43(2-3):85–96, 2003.

[43] Christopher Crick, Graylin Jay, Sarah Osentoski, and Odest Chadwicke Jenkins. Ros and rosbridge: roboticists out of the loop. In Holly A. Yanco, Aaron Steinfeld, Vanessa Evers, and Odest Chadwicke Jenkins, editors, *HRI*, pages 493–494. ACM. ISBN 978-1-4503-1063-5.

[44] Alexander Clifford Perzylo M.J.G. (Ren) Van de Molengraft Daniel Di Marco, Rob Janssen and Paul Levi. A deliberation layer for instantiating robot execution plans from abstract task descriptions. In *Workshop on Planning and Robotics 23rd International Conference on Automated Planning and Scheduling*, 2013.

[45] Giuseppe de Giacomo, Yves Lespérance, and Hector J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artif. Intell.*, 121(1-2):109–169, August 2000. ISSN 0004-3702. doi: 10.1016/S0004-3702(00)00031-X.

[46] Tinne De Laet. *Rigorously Bayesian Multitarget Tracking and Localization*. PhD thesis, May 2010.

[47] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, page 13, 2004.

[48] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[49] J. Ruiz del Solar, E. Chown, and P. Ploeger, editors. *RoboCup 2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes In Artifical Intelligence*. Springer Verlag Berlin Heidelberg, 2011. ISBN 978-3-642-20216-2.

[50] Micheal Delp. Experiments in off-policy reinforcement learning with the gq(lambda) algorithm. In *PhD Thesis*.

[51] Saadia Dhouib, Nicolas Du Lac, Jean-Loup Farges, Sébastien Gerard, Miniar Hemaissia-Jeannin, Juan Lahera-Perez, Stéphane Millet, Bruno Patin, and Serge Stinckwich. Control Architecture Concepts and Properties of an Ontology Devoted to Exchanges in Mobile Robotics. In *6th National Conference on Control Architectures of Robots*, page 24 p., Grenoble, France, May 2011. INRIA Grenoble Rhône-Alpes.

[52] R.O. Duda and P.E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972. ISSN 0001-0782.

[53] Daniel Elenius, Grit Denker, David Martin, Fred Gilham, John Khouri, Shahin Sadaati, and Rukman Senanayake. The owl-s editor - a development tool for semantic web services. In Asuncin Gmez-Prez and Jrme Euzenat, editors, *ESWC*, Lecture Notes in Computer Science, pages 78–92. Springer. ISBN 3-540-26124-9.

[54] J. Elfring, S. Van Den Dries, R. van de Molengraft, and M. Steinbuch. Semantic world modeling using probabilistic multiple hypothesis anchoring. *Robot. Auton. Syst.*, 61(2):95–105, February 2013. ISSN 0921-8890.

[55] Kutluhan Erol. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation*. PhD thesis, College Park, MD, USA, 1996. UMI Order No. GAX96-22054.

[56] R.E. Fikes and N.J. Nilsson. strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

[57] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. Technical Report 43R, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, May 1971. SRI Project 8259.

[58] R.J. Firby. An investigation into reactive planning in complex domains. In *Proc. of the Sixth National Conference on Artificial Intelligence*, volume 1, pages 202–206, 1987.

[59] Florian Fischer, Gulay nel, Barry Bishop, and Dieter Fensel. Towards a scalable, pragmatic knowledge representation language for the web. In Amir Pnueli, Irina Virbitskaite, and Andrei Voronkov, editors, *Ershov Memorial Conference*, volume 5947 of *Lecture Notes in Computer Science*, pages 124–134. Springer, 2009. ISBN 978-3-642-11485-4.

[60] B. Gerkey, R. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, June .

[61] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language.

[62] Giuseppe Giacomo, Yves Lespérance, Hector J. Levesque, and Sebastian Sardina. IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents. pages 31–72. 2009. doi: 10.1007/978-0-387-89299-3_2.

[63] E. Guizzo. Three engineers, hundreds of robots, one warehouse. *Spectrum, IEEE*, 45(7):26 –34, july 2008.

[64] Young-Guk Ha, Joo-Chan Sohn, Young-Jo Cho, and Hyunsoo Yoon. A robotic service framework supporting automated integration of ubiquitous sensors and devices. *Inf. Sci.*, 177(3):657–679, 2007.

[65] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 1558609016.

[66] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. ISSN 0536-1567.

[67] Ronny Hartanto. *A Hybrid Deliberative Layer for Robotic Agents - Fusing DL Reasoning with HTN Planning in Autonomous Robots*, volume 6798 of *Lecture Notes in Computer Science*. Springer, 2011. ISBN 978-3-642-22579-6.

[68] T. Hester and P. Stone. Negative information and line observations for monte carlo localization. In *ICRA*, pages 2764–2769, 2008.

[69] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *CoRR*, abs/1106.0675, 2011.

[70] Berthold K. Horn. *Robot Vision*. McGraw-Hill Higher Education, 1st edition, 1986. ISBN 0070303495.

[71] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium.

[72] Dominique Hunziker, Mohanarajah Gajamohan, Markus Waibel, and Raffaello DAndrea. Rapyuta: The RoboEarth cloud engine. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), Karlsruhe, Germany*, pages 438–444, 2013.

[73] S. Hutchinson, G. Hager, and P. Corke. A tutorial on visual servo control. *IEEE Trans. on Robotics and Automation*, 12(5):651–670, October 1996.

[74] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[75] J. Illingworth and J. Kittler. A survey of the hough transform. *Computer Vision, Graphics, and Image Processing*, 44(1):87–116, 1988. ISSN 0734-189X.

[76] J. Jackson. Microsoft robotics studio: A technical introduction. *Robotics Automation Magazine, IEEE*, 14(4):82 –87, dec. 2007. ISSN 1070-9932. doi: 10.1109/M-RA.2007.905745.

[77] Mohammad Jamshidi. Autonomous control of complex systems: robotic applications. *Applied Mathematics and Computation*, 120(1-3):15–29, 10 May 2001. doi: doi:10.1016/S0096-3003(99)00285-4.

[78] R. Janssen, J. de Best, R. van de Molengraft, and M. Steinbuch. The design of a semi-automated football table. In *IEEE International Conference on Control Applications (CCA)*, pages 89 –94, sept. 2010.

[79] Rob Janssen, Jeroen de Best, and René van de Molengraft. Real-time ball tracking in a semi-automated foosball table. In *RoboCup*, pages 128–139, 2009.

[80] Rob Janssen, Mark Verrijt, Jeroen de Best, and René van de Molengraft. Ball localization and tracking in a highly dynamic table soccer environment. *Mechatronics*, 22(4):503–514, June 2012. ISSN 09574158.

[81] N. Jong and P. Stone. Hierarchical model-based reinforcement learning: Rmax + MAXQ.

[82] M. Jovanovic and D. Starcevic. Software architecture for ground control station for unmanned aerial vehicle. In *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*, pages 284–288, 2008. ISBN 978-0-7695-3114-4.

[83] Alex Juarez, Christoph Bartneck, and Loe Feijs. Using semantic technologies to describe robotic embodiments. In *Proceedings of the 6th International Conference on Human-robot Interaction*, HRI '11, pages 425–432, 2011. ISBN 978-1-4503-0561-7.

[84] D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, 1 edition. ISBN 0130950696.

[85] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, May 1998. ISSN 0004-3702. doi: 10.1016/S0004-3702(98)00023-X.

[86] H. Kälviäinen, P. Hirvonen, L. Xu, and E. Oja. Probabilistic and non probabilistic hough transforms: overview and comparisons. *Image and Vision Computing*, 13 (4):239–252, 1995. ISSN 0262-8856.

[87] Ryan Francis Kelly. *Asynchronous Multi-Agent Reasoning in the Situation Calculus*. Phd, The University of Melbourne, 2008.

[88] Jong-Hwan Kim, In-Bae Jeong, In-Won Park, and Kang-Hee Lee. Multi-layer architecture of ubiquitous robot system for integrated services. *I. J. Social Robotics*, 1(1):19–28, 2009.

[89] T. Kirubarajan and Y. Bar-Shalom. Kalman filter versus imm estimator: when do we need the latter? *Aerospace and Electronic Systems, IEEE Transactions on*, 39 (4):1452 – 1457, oct. 2003. ISSN 0018-9251.

[90] N. Kiryati, H. Kälviäinen, and S. Alaoutinen. Randomized or probabilistic hough transform: unified performance evaluation. *Pattern Recognition Letters*, 21(13-14):1157–1164, 2000. ISSN 0167-8655.

[91] M. Klusch and A. Gerber. Semantic web service composition planning with owls-xplan. In *In Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, pages 55–62, 2005.

[92] J. Kober, A. Wilhelm, E. Oztop, and J. Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. (4):361–379, 2012.

[93] Jens Kober, Katharina Mlling, Oliver Kromer, Christoph H. Lampert, Bernhard Scholkopf, and Jan Peters. Movement templates for learning of hitting and batting.

[94] Jens Kober, Erhan Oztop, and Jan Peters. Reinforcement learning to adjust robot movements to new situations. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2650–2655, 2011. ISBN 978-1-57735-515-1.

[95] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, 2004.

[96] J. Zico Kolter and Andrew Y. Ng. Learning omnidirectional path following using dimensionality reduction. In Wolfram Burgard, Oliver Brock, and Cyrill Stachniss, editors, *Robotics: Science and Systems*. The MIT Press, 2007. ISBN 978-0-262-52484-1.

[97] O. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. (9):1105–1116, 2010.

[98] Torsten Kröger and Friedrich M Wahl. Online trajectory generation: basic concepts for instantaneous reactions to unforeseen events. *IEEE Transactions on Robotics*, 26(1):94–111, 2010.

[99] L. Kunze, T. Roehm, and M. Beetz. Towards semantic robot description languages. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5589 –5595, may 2011. doi: 10.1109/ICRA.2011.5980170.

[100] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991. ISBN 079239206X.

[101] V. F. Leavers. Which hough transform? *CVGIP: Image Understanding*, 58(2): 250–264, 1993. ISSN 1049-9660.

[102] M. Lekavy and P. Návrat. *Expressivity of STRIPS-Like and HTN-Like Planning*, volume 4496/2007 of *Lecture Notes in Computer Science*, pages 121–130. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72829-0. doi: 10.1007/978-3-540-72830-6_13.

[103] H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R.B. Scherl. Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1-3):59–83, 1997.

[104] X.R. Li and V.P. Jilkov. A survey of maneuvering target tracking–Part III: Measurement models. In *Proceedings of the 2001 SPIE Conference on Signal and Data Processing of Small Targets*, volume 4473, pages 423–446, 2001.

[105] X.R. Li and V.P. Jilkov. A survey of maneuvering target tracking–Part IV: Decision-based methods. In *Proceedings of the 2002 SPIE Conference on Signal and Data Processing of Small Targets*, volume 4728, pages 511–534, 2002.

[106] X.R. Li and V.P. Jilkov. Survey of maneuvering target tracking. Part I: Dynamic models. *Aerospace and Electronic Systems, IEEE Transactions on*, 39(4):1333 – 1364, oct. 2003. ISSN 0018-9251.

[107] X.R. Li and V.P. Jilkov. A survey of maneuvering target tracking: Approximation techniques for nonlinear filtering. In *Proceedings of the 2004 SPIE Conference on Signal and Data Processing of Small Targets*, volume 5428, pages 537–550, 2004.

[108] X.R. Li and V.P. Jilkov. Survey of maneuvering target tracking. Part V: Multiple-model methods. *Aerospace and Electronic Systems, IEEE Transactions on*, 41(4): 1255 – 1321, oct. 2005. ISSN 0018-9251.

[109] X.R. Li and V.P. Jilkov. A survey of maneuvering target tracking–Part VIa: Density-based exact nonlinear filtering. In *Proceedings of the 2010 SPIE Conference on Signal and Data Processing of Small Targets*, volume 7698, 2010.

[110] X.R. Li and V.P. Jilkov. A survey of maneuvering target tracking–Part VIb: Approximate nonlinear density filtering in mixed time. In *Proceedings of the 2010 SPIE Conference on Signal and Data Processing of Small Targets*, volume 7698, 2010.

[111] X.R. Li and V.P. Jilkov. Survey of maneuvering target tracking. Part II: Motion models of ballistic and space targets. *Aerospace and Electronic Systems, IEEE Transactions on*, 46(1):96 –119, jan. 2010. ISSN 0018-9251.

[112] X.R. Li and V.P. Jilkov. A survey of maneuvering target tracking–Part VIc: Approximate nonlinear density filtering in discrete time. In *Proceedings of the 2011 SPIE Conference on Signal and Data Processing of Small Targets*, volume 8137, 2011.

[113] X.R. Li and V.P. Jilkov. A survey of maneuvering target tracking–Part VId: Sampling based nonlinear filtering. In *Proceedings of the 2011 SPIE Conference on Signal and Data Processing of Small Targets*, volume 8137, 2011.

[114] D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res. (JAIR)*, 20:1–59, 2003.

[115] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *ICRA*, pages 1642–1648, 2010.

[116] Hamid Reza Maei. *Gradient Temporal-Difference Learning Algorithms*. PhD thesis, University of Alberta, 2011.

[117] Hamid Reza Maei, Csaba Szepesvári, Shalabh Bhatnagar, and Richard S. Sutton. Toward off-policy learning control with function approximation. In *ICML*, pages 719–726, 2010.

[118] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-13360-1.

[119] D. Di Marco, M. Tenorth, K. Häussermann, O. Zweigle, and P. Levi. Roboearth action recipe execution. In *12th International Conference on Intelligent Autonomous Systems*, 2012.

[120] Daniel Di Marco, Moritz Tenorth, Kai Hussermann, Oliver Zweigle, and Paul Levi. Roboearth action recipe execution. In Sukhan Lee, Kwang-Joon Yoon, and Jangmyung Lee, editors, *Frontiers of Intelligent Autonomous Systems*, Studies in Computational Intelligence, pages 117–126. Springer. ISBN 978-3-642-35484-7.

[121] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, et al. Bringing semantics to web services: The owl-s approach. *Semantic Web Services and Web Process Composition*, pages 26–42, 2005.

[122] T. Martinez-Marin and R. Rodriguez. Navigation of autonomous vehicles in unknown environments using reinforcement learning. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 872–876, 2007. doi: 10.1109/IVS.2007.4290226.

[123] E. Mazor, A. Averbuch, Y. Bar-Shalom, and J. Dayan. Interacting multiple model methods in target tracking: a survey. *Aerospace and Electronic Systems, IEEE Transactions on*, 34(1):103 –123, jan 1998.

[124] John McCarthy. Situations, actions, and causal laws. Technical Report Memo 2, Stanford Artificial Intelligence Project, Stanford University, 1983.

[125] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[126] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl-the planning domain definition language. *The AIPS-98 Planning Competition Comitee*, 1998.

[127] C. McGann, E. Berger, J. Bohren, S. Chitta, B. Gerkey, S. Glaser, B. Marthi, W. Meeussen, T. Pratkanis, E. Marder-Eppstein, and M. Wise. Model-based, hierarchical control of a mobile manipulation platform. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, Thessaloniki, Greece, 2009.

[128] Francisco S. Melo and M. Isabel Ribeiro. Q -learning with linear function approximation. In Nader H. Bshouty and Claudio Gentile, editors, *COLT*, volume 4539 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 2007. ISBN 978-3-540-72925-9.

[129] O. Michel. Webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):39–42, 2004.

[130] *1/2-Inch VGA (With Freeze-Frame) Cmos Active-Pixel Digital Image Sensor, MT9V403 Data Sheet*. Micron Technology, Inc, 2004. Document number: 09005aef80c07280, Micron Part Number: MT9V403C12ST.

[131] Nick Milton, David Clarke, and Nigel Shadbolt. Knowledge engineering and psychology: Towards a closer relationship. *International Journal of Human-Computer Studies*, 64(12):1214 – 1229, 2006. ISSN 1071-5819. doi: http://dx.doi.org/10.1016/j.ijhcs.2006.08.001.

[132] Jos Mira, Jos Ramn lvarez Snchez, and Flix de la Paz. The knowledge engineering approach to autonomous robotics. In Jos Mira and Jos R. lvarez, editors, *IWANN (2)*, volume 2687 of *Lecture Notes in Computer Science*, pages 161–168. Springer, 2003. ISBN 3-540-40211-X.

[133] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.

[134] L. Mösenlechner and M. Beetz. Fast temporal projection using accurate physics-based geometric reasoning. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 6–10 2013. Accepted for publication.

[135] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. In *Proceedings of the AAAI 2012 Fall Symposium on robots that Learn Interactively from Human Teachers*, 2012.

[136] A. Myrup and M. Ording-Thomsen. Automated foosball table, 2007.

[137] D. Nau, T.C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, and F. Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20 (1):379–404, 2003.

[138] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558608567.

[139] I.A. Nesnas, A.W., M. Bajracharya, R. Simmons, T. Estlin, and W.S. Kim. Claraty: An architecture for reusable robotic software. In *SPIE Aerosense Conference*, 2003.

[140] Tim Niemueller, Gerhard Lakemeyer, and Siddhartha S. Srinivasa. A Generic Robot Database and its Application in Fault Analysis and Performance Evaluation. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems 2012*, Vilamoura, Algarve, Portugal, 2012. IEEE/RAS.

[141] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009.

[142] Christopher Painter-Wakefield. *Sparse Value Function Approximation for Reinforcement Learing*. PhD thesis, Duke University, 2013.

[143] Rajendra Patel, Mikael Hedelind, and Pablo Lozan-Villegas. Enabling robots in small-part assembly lines: The "rosetta approach" - an industrial perspective. In *ROBOTIK*. VDE-Verlag, 2012. ISBN 978-3-8007-3418-4.

[144] R.S. Pieters. Active Vision: Directing Visual Attention. Master's thesis, Eindhoven University of Technology, 2008.

[145] *Prosilica GC640/640c Data Sheet*. Prosilica Inc, 2008.

[146] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[147] Matthias Radestock and Susan Eisenbach. Coordinating components in middleware systems. *Concurrency and Computation: Practice and Experience*, 15(13): 1205–1231, 2003.

[148] Vignesh Ramanathan and Axel Pinz. Active object categorization on a humanoid robot. In Leonid Mestetskiy and Jos Braz, editors, *VISAPP*, pages 235–241. SciTePress, 2011. ISBN 978-989-8425-47-8.

[149] CH.H. Reinsch. Smoothing by spline functions. *Numerische Mathematik*, pages 177–183.

[150] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, Massachusetts, MA, illustrated edition edition, 2001. ISBN 0262182181.

[151] Warren L. Rhodes. Color separation techniques. *Color Research And Application*, 5(2):123–123, 1980. ISSN 1520-6378.

[152] Leonard Richardson and Sam Ruby. Restful web services.

[153] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010. ISBN 978-0-13-207148-2.

[154] Conrad Sanderson, Ryan Curtin, Ian Cullinan, Dimitrios Bouzas, and Stanislav Funiak. Armadillo: C++ linear algebra library , August 2013.

[155] Dan Sanderson. *Programming Google App Engine - Build and Run Scalable Web Apps on Google's Infrastructure.* O'Reilly, 2010. ISBN 978-0-596-52272-8.

[156] S.R. Schmidt-Rohr, F. Romahn, P. Meissner, R. Jäkel, and R. Dillmann. Learning probabilistic decision making by a service robot with generalization of user demonstrations and interactive refinement. In *Frontiers of Intelligent Autonomous Systems*, pages 309–322. Springer, 2013.

[157] John Schulman, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. *Submitted. Draft at https://sites. google. com/site/rss2013trajopt*, 2013.

[158] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004. ISBN 026219502X.

[159] M. Silbert, S. Sarkani, and T. Mazzuchi. Comparing the state estimates of a kalman filter to a perfect imm against a maneuvering target. In *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on*, pages 1–5, july 2011.

[160] R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1931–1937, 1998.

[161] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. Htn planning for web service composition using shop2. *Web Semant.*, 1(4):377–396, October 2004. ISSN 1570-8268. doi: 10.1016/j.websem.2004.06.005.

[162] Russel Smith. Open dynamics engine, May 2007.

[163] Peter Stone, Richard S. Sutton, and Satinder P. Singh. Reinforcement learning for 3 vs. 2 keepaway. In *RoboCup 2000: Robot Soccer World Cup IV*, pages 249–258, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42185-8.

[164] S. Sumino, A. Mutoh, and S. Kato. Evolutionary approach of reward function for reinforcement learning using genetic programming. In *Micro-NanoMechatronics and Human Science (MHS), 2011 International Symposium on*, pages 385–390, 2011. doi: 10.1109/MHS.2011.6102214.

[165] Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[166] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction, 1998.

[167] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[168] Richard S. Sutton, Satinder Singh, Doina Precup, and Balaraman Ravindran. Improved switching among temporally abstract actions. In *Advances in Neural Information Processing Systems 11*, pages 1066–1072. MIT Press, 1999.

[169] Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, pages 761–768, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0-9826571-6-1, 978-0-9826571-6-4.

[170] S. Suzuki and K. Be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1): 32–46, April 1985. ISSN 0734189X.

[171] D. Sýkora, D. Sedláček, and K. Riege. Real-time color ball tracking for augmented reality. In *Proceedings of the 14th Eurographics Symposium on Virtual Environments*, pages 9–16, May 2008.

[172] S. Takata and T. Hirano. Human and robot allocation method for hybrid assembly systems. *CIRP Annals - Manufacturing Technology*, 60(1):9 – 12, 2011.

[173] M. Tenorth and M. Beetz. Knowrob – knowledge processing for autonomous personal robots. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4261–4266. IEEE, 2009.

[174] M. Tenorth and M. Beetz. Knowledge processing for autonomous robot control. In *AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI*, Stanford, CA, USA, March 26–28 2012.

[175] M. Tenorth, A. Perzylo, R. Lafrenz, and M. Beetz. The RoboEarth language: Representing and Exchanging Knowledge about Actions, Objects, and Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, USA, 2012. Best Cognitive Robotics Paper Award.

[176] Moritz Tenorth, Fernando De la Torre, and Michael Beetz. Learning probability distributions over partially-ordered human everyday activities. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 6–10 2013. Accepted for publication.

[177] Moritz Tenorth, Alexander Clifford Perzylo, Reinhard Lafrenz, and Michael Beetz. Representation and Exchange of Knowledge about Actions, Objects, and Environments in the RoboEarth Framework. *IEEE Transactions on Automation Science and Engineering (T-ASE)*, 2013. Accepted for publication.

[178] The MathWorks, Inc. Matlab/simulin, August 2013.

[179] A. M. Turing. Computing machinery and intelligence. 59(236):433–460, October 1950. ISSN 0026-4423.

[180] T. van der Zant and T. Wisspeintner. Robocup x: A proposal for a new league where robocup goes real world. *RoboCup 2005: Robot Soccer World Cup IX*, pages 166–172, 2006.

[181] J. Vergés-Llahí. Color constancy and image segmentation techniques for applications to mobile robotics. Doctoral thesis, Universitat Politècnica de Catalunya, 2005.

[182] V. Verma, T. Estlin, A. Jónsson, C. Pasareanu, R. Simmons, and K. Tso. Plan execution interchange language (plexil) for executable plans and command sequences. In *Intl. Symp. on Artificial Intelligence, Robotics and Automation in Space (ISAIRAS), Germany*, 2005.

[183] L.F.M. Vieira, Uichin Lee, and M. Gerla. Phero-trail: a bio-inspired location service for mobile underwater sensor networks. *IEEE Journal on Selected Areas in Communications*, 28(4):553 –563, may 2010.

[184] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J.M.M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft. Roboearth. *Robotics Automation Magazine, IEEE*, 18(2):69–82, 2011. ISSN 1070-9932. doi: 10.1109/MRA.2011.941632.

[185] Vanessa Wang, Frank Salim, and Peter Moskovits. *The Definitive Guide to HTML5 WebSocket*. Apress, Berkely, CA, USA, 1st edition, 2013. ISBN 1430247401, 9781430247401.

[186] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[187] T. Weigel. Kiro: A table soccer robot ready for the market. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4266 – 4271, april 2005.

[188] T. Weigel and B. Nebel. Starkick, 2003.

[189] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA, 1995.

[190] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009. ISBN 0596521979, 9780596521974.

[191] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. ISSN 1471-0684.

[192] Ronald J Williams and Leemon C Baird. Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems. *Technical Rep. NU-CCS-93-11, Boston*, 1993.

[193] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38, December 2006. ISSN 0360-0300.

[194] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transform methods for circle finding. *Image and Vision Computing*, 8(1):71–77, 1990. ISSN 0262-8856.

[195] Xianrong Zheng and Yuhong Yan. An efficient syntactic web service composition algorithm based on the planning graph model. In *Proceedings of the 2008 IEEE International Conference on Web Services*, ICWS '08, pages 691–699, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3310-0. doi: 10.1109/ICWS.2008.134.

Appendix A

Analytical Time Delay Estimation

The total time delay is a result of individual processing steps. These steps will be reviewed here with respect to the parameters currently used on the setup. That is, the camera runs at $f_{vision} = 200$ Hz capturing 8-bit monochrome images of 657×446 pixels.

Exposure Time

Although the exposure time can be set separately, it is desired to be as long as possible at high frame-rates. The delay it causes will therefore be

$$\tau_{et} = \frac{1}{f_{vision}} \tag{A.1}$$

Therefore $\tau_{et} = 5$ ms.

Sensor Readout Time

The Prosilica GC640c sensor type is a 1/2" CMOS progressive scan MT9V403 [145]. It can capture an active resolution of 493 rows by 659 columns and has a clock speed (f_{clock}) of 66 MHz. Windowing can be used to reduce the capture resolution. The number of clock cycles for reading each row ($c_{row} = 671$ cycles) however, is independent of how

many columns (n_{col}) are captured [130]. The delay can be calculated with

$$\tau_{srt} = n_{row} \frac{c_{row}}{f_{clock}} \tag{A.2}$$

where n_{row} is the number of rows captured with the window. At full resolution $\tau_{srt} =$ 4.9 ms.

Data Transfer Time

The captured data is sent from the camera to the computer via Gigabit ethernet. Assuming sending is not started before the sensor has been fully read, the maximum delay for this operation is

$$\tau_{dtt} = \frac{n_{bit} n_{row} n_{col}}{1e9} \tag{A.3}$$

where n_{bit} is 8 bit for monochrome images. Therefore τ_{dtt} = 2.6 [ms]

Vision Processing

The amount of time it takes to process one image is less or equal to the sample time.

$$\tau_{vp} \leq \frac{1}{f_{vision}} \tag{A.4}$$

which therefore is 5 [ms].

Therefore, the total worst case delay due to the complete image processing pipe line is 17.5 [ms].

Dankwoord

A wise man often tells me that moments in life are best experienced when one pauses for a second, when one becomes at ease, when one takes the time to be proud of who we are and where we come from, and how important it is to fully enjoy, that what is near us. For me this is one of these moments, the moment that I get to thank the people that have contributed to who I am, and for where they have brought me.

I first of all would like to thank my professor, Maarten Steinbuch, for granting me a place in his group. During my PhD, as well as during my master and bachelor program, I have always received the exact amount of guidance and support that was needed to complete my goals, but Maarten has also always given me the right amount of freedom, to explore new paths with an own sense of creativity. I also want to thank Maarten for the financial support our group has always received, that allowed us to visit conferences, develop new platforms and work with the most state of the art equipment.

I want to thank René van de Molengraft, my supervisor, for the incredible amount of support that I have always received, both professionally and personally, and for offering me a possibility to join in on the RoboEarth project. I want to thank Rene for the vast amount of hours that we spent in room -1.141, brainstorming about my project, always with an open, free and unbound mind. I especially want to thank René for the incredible amount of trust he has always had in me, and for the enlightening advice and inspiring 'pep-talks', whenever my research became difficult or hard to oversee.

I want to thank the members of my committee, for evaluating my work and proofreading my thesis. I would like to personally thank Herman Bruyninckx and Raffaello D'Andrea, for their additional support during my PhD, and for showing me the industrial opportunities and societal impact of robotics research.

I furthermore want to thank my colleagues at Eindhoven University. Especially Rob Hoogendijk, in which I always had both friend and teammate, in all of our travels, tournaments and RoboCup achievements. I want to thank Janno Lunenburg, Sjoerd van den Dries and Jos Elfring, for they have been the best possible cooperatives in providing support, advice and reflection. The achievements described in this work could not have been accomplished without them. I also want to thank my students, that provided me with the huge amounts of work required for implementations and experiments, and for the many

hours we discussed over existing concepts and the possibilities for new advancements.

I want to thank all of my friends, that made me who I am as a person and allowed me to have an incredible life outside my work.

I want to thank my family, and especially my father, mother and brothers, who have always made me feel comfortable with my own self, who made me strong as a person and who always made me laugh. They are me, and I am them.

Finally, I want to thank my wife, Allison, my everlasting support and true best friend, in all adventures that we made thus far and will make, voor nu en voor altijd.

Curriculum Vitae

Rob Josephus Maria Janssen was born on December 26th 1980 in Eindhoven, The Netherlands. After completing the Mechanical Engineering Bachelor program of Eindhoven University of Technology, he started their subsequent Master program in 2006. During this period, Rob participated in the Eindhoven University Tech United robot soccer team, where he worked on data association and opponent modeling techniques. As part of his studies, Rob remained at the University of Florida for 4 months as an intern, where he worked on vision based state estimation techniques for unmanned aircraft. His master thesis was based on the design and implementation of vision based control techniques applied to the Eindhoven University automated football table. Rob completed the Mechanical Engineering Master program in August 2009 (Cum Laude).

In September 2009, Rob started his PhD at Eindhoven University of Technology, as a member of the European funded RoboEarth project. His research topics included A.I. based learning and planning techniques, logical reasoning and the design of robot task execution architectures. He furthermore completed the D.I.S.C. program in 2010, and was an active member of both the Tech United robot soccer and RoboCup@Home teams. During his PhD he visited ETH Zürich for a 4 month period, to cooperate with ETH PhDs on the development of learning controllers applied within the RoboEarth project. Rob defended his PhD on Tuesday, April 15th 2014.