# LEARNING BY OBSERVATION USING QUALITATIVE SPATIAL RELATIONS

by

JAY YOUNG

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
September 2015

**Acknowledgements**

My gratitude extends firstly and to none moreso than my supervisor Dr. Nick Hawes, whose seemingly endless enthusiasm, patience and support have been invaluable. While I have learned a great deal many new technical skills over my period of study, I have learned equally how to be an effective researcher, academic and teacher from my supervisor, and I would not have completed this work without his feedback and guidance, usually provided with impeccable timing. While a PhD is a serious undertaking in and of itself, there are a huge number of things that need to be learned concurrently that are not advertised as part of the syllabus, and a less generous supervisor would not have taken the time to teach those things.

Though the front cover of this thesis bears my name alone, I am without question a product of the School of Computer Science at the University of Birmingham and its community of scholars who, in one small (or large!) way or another, have all contributed to this thesis. My thanks extends to all of those that gladly shared their knowledge and advice with me over the years to prepare me with the tools I needed to complete my BSc, to then undertake an MSc degree, and then finally a PhD. I wish to extend special thanks to Professor Jon Rowe who expertly introduced me to a range of techniques, skills and ways of thinking that altered the way I approached problem solving in Computer Science. Further specific thanks goes to Dr. Peter Hancox, Professor Jeremy Wyatt, and Dr. Christine Zarges, all of whom have been members of my Thesis Group at some point and whose feedback has contributed to my work. Additional thanks to Professor Benjamin Kuipers and Professor Aaron Sloman. Now that I am supervising students of my own, my primary goal is to give them the same kind of support that was given to me.

Thanks also go to Hannah for being so supportive and putting up with me.

**Peer-reviewed publications arising from this work**

1. **Jay Young** and Nick Hawes. *"Learning by Observation Using Qualitative Spatial Relations"*. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS).* **2015**.

2. **Jay Young** and Nick Hawes. *"Learning Micro-Management Skills in RTS Games by Imitating Experts"*. In *Proceedings of the 10th Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.* **2014**.

3. **Jay Young** and Nick Hawes. *"Effects of Training Data Variation and Temporal Representation in a QSR-Based Action Prediction System"*. In *Proceedings of the AAAI 2014 Spring Symposium on Qualitative Representations for Robots.* **2014**.

4. **Jay Young** and Nick Hawes. *"Predicting Situated Behaviour Using Sequences Of Abstract Spatial Relations"*. In *Proceedings of the AAAI 2013 Fall Symposium "How Should Intelligence be Abstracted in AI Research: MDPs, Symbolic Representations, Artificial Neural Networks, or ?"*. **2013**.

5. **Jay Young**, Fran Smith, Christopher Atkinson, Ken Poyner, Nick Hawes and Tom Chothia. *"SCAIL: An integrated Starcraft AI System"*. In *Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games.* **2012**.

**Abstract**

In recent years much work in Artificial Intelligence has focused on the problem of Learning By Observation, a kind of learning task where an agent learns how to act in a particular domain through the observation of an expert and replication of their behaviour. In this work, we investigate a specific formulation of the problem where traces of expert behaviour are considered to be limited in availability, noisy, and observed from an external viewpoint. We are particularly interested in how knowledge about the world is delivered to the learning agent, and we present a system for Learning By Observation using Qualitative Spatial Relations as a method of knowledge delivery. We show how our work with QSRs allows us to represent noisy, metric observations about the world as abstract symbols, and compactly represent spatio-relational features that the original, metric representation does not have. We present a technique for searching the parameter space that controls the degree of these abstractions, so that specific spatial features can be discovered that provide higher degrees of predictive power regarding the actions chosen by an observed expert in a particular state. Our work is evaluated in two spatially-situated domains that feature adversarial and co-operative interactions. First, RoboCup Soccer a multi-agent simulation of Football, where agents are subject to the robotic paradigms of noisy observations, partial observability and uncertain actions. Second, Starcraft, a popular Real-Time Strategy game, where control of units requires real-time input to control large numbers of agents. Our experiments are divided into two sections. First we show how our pipeline is able to mimic the behaviour of agents – both synthetic and human – in these domains, and copy their behaviour to a high degree of accuracy. Second, we show how our expert models can be used to modify the action selection policy of a Reinforcement Learning algorithm by giving the algorithm the option of taking actions that an expert would take in a particular state. This reduces the amount of state-space exploration the algorithm needs to do, and provides overall better initial results. Interestingly, the algorithm is able to improve upon the underlying expert behaviour, eventually significantly out-performing the expert on every task.

# Contents

# List of Figures

# List of Algorithms

# Introduction

Engineering AI systems is a non-trivial task. Systems that can *learn* solutions to problems are therefore desirable, and in some cases, learning AI systems may be able to display better performance than if they were being programmed by a human developer. Learning algorithms are well-known for discovering novel and surprising solutions to problems, and uncovering information that human developers overlook.

From a technical standpoint, learning-by-doing techniques (e.g. reinforcement learning) can be risky due to the need for trial-and-error, which can be dangerous and expensive in certain domains (such as robotics), and often requires the design of reward functions, which can be non-trivial. Alternatively, if expert agents already exist for a target domain, then an AI system could learn to act within that domain by itself through observing those experts, . Humans do this all the time from the youngest age in a process known as *Learning by Observation* (LbO) [55], which can take the form of passive, non-interactive learning. One example of where this is commonly seen is when humans learn to play a new sport – they may initially observe the technique of expert players and attempt to imitate them in order to achieve an initial, baseline level of performance before seeking further improvement and learning of advanced techniques [25].

1. The imitated behavior is new for the imitator,

2. The same task strategy as that of the demonstrator is employed.

3. The same task goal is accomplished.

In Learning by Demonstration, an expert may explicitly demonstrate a technique to a learner,

who may be able to ask questions or ask for lessons to be repeated or information clarified . In our work we are interested in the passive case, where no interaction or communication between expert and learner is available, and under the constraint there exists a finite amount of expert data to learn from.

The constraint of limited expert data increases the difficulty of the learning task, and provides no guarantee as to how comprehensively the expert will demonstrate the range of activities possible in the task domain. This means that we must study how a system may best extract information from that limited source of data, how it *makes use* of that data, how it can deal with the gaps in knowledge that will naturally occur from data acquisition under these constraints, and how the system will transition from one form of learning to another as it approaches the limits of what it can learn purely from observation.

These sorts of problems have been approached in many different ways, such as through data-mining and pre-processing techniques to generalise and adapt examples to new situations. In our work instead we begin to address these problems from the starting point of *knowledge representation*, with the first thought that when designing any learning system, we must consider how it will represent its environment, and what features that representation will provide to the system in terms of the properties of entities it observes, and itself. *Quantitative representations*, i.e. representations which use real numbers (to measure, for example, position, area, time, velocity etc.) are highly precise, but can be brittle when a system must generalise information. Uncertain observations (due to noisy sensors or dynamic environments) exacerbate this problem, and states that would appear to be semantically identical to a human observer may be radically different when using this kind of raw representation.

Much like many learning tasks, LbO tasks are characterised by access to *limited* amounts of expert data. One of the things we are interested in is how the ways the *knowledge representation* of this data can make it more useful, by which we mean both in terms of generalisation and also in extracting and revealing structure that might exist within it. In our work we approach the problem of LbO by using *Qualitative Spatial Relations* (QSRs) to represent observations of the continuous, spatially-situated domains in which we learn, predict and ultimately replicate

the behaviour of experts. We use qualitative representations to discretise over the continuous variables which typically describe the behaviour of mobile entities, such as position, orientation and velocity, replacing them instead with symbolic representations, where these symbols represent ranges of possible continuous values. These techniques may also be *relational*, encoding information about relative properties that hold between entities at a given instant [30]. Our hypothesis is that encoding of relational information provides additional explanatory power in spatially-situated, co-operative or adversarial domains where decision making is typically based on relationships between entities, and so these are the kinds of domains in which we evaluate our work.

Consider a simple scenario from Soccer – that of a player attempting to shoot a ball into a goal which is defended by a goalkeeper, as shown in Figure 1.1. When shooting at the goal, the *relative* angles between the player, goalkeeper and goal posts are important. The goalkeeper would like to minimise the angle between himself and a goalpost with respect to the player, so as to make the shot more difficult for the opponent by reducing the amount of goal area that can be reached, whereas the player would prefer to widen the angle, to open up more of the area of the goal for potential shots, and make the shot easier. As there may be a range of angles that are satisfactory to either player, decision making in this task may be interpreted in a qualitative, relational context, with action selection being dependent on the relations that hold in the scene at some critical instant in time. The player may have taken other information into account when leading up to this state, such as the angles, velocities and positions of opposing defending players.

Spatial-relational information is crucially important in describing and understanding this particular scenario, and thus mimicking the behaviour of either the player or the goalkeeper. But it would not be immediately available in any meaningful form if we observed the raw metric data representing the sequence of events described. It may be possible to discover, with some pre-processing, that such relations exist and are important, but it would be more convenient and useful to have this information to-hand in the state representation already. The next question that this raises is how to encode this relational data without knowing beforehand what information

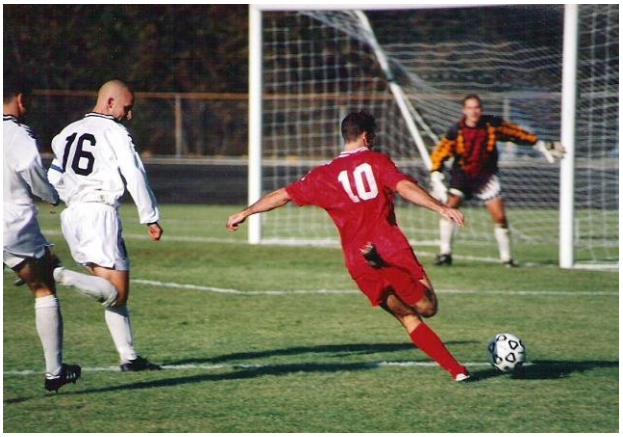

Figure 1.1: An example of a goalkeeper attempting to minimise the angle between himself and the goalpost, with respect to the forward player. *Credit Rick Dikeman, 1996, licensed under Creative Commons BY-SA 3.0.*

is important, and to what degree of granularity any qualitative abstraction should be applied. It would be far better to be able to determine what is required automatically, by learning what information is important to the experts in their action selection. In this way, rather than devoting pre-processing time to discovering important relational information and generalising the data, we might instead devote that time to *improving* and *refining* generalisations that already exist. This is what we achieve in this work.

## 1.1 Objectives and Contributions of this Thesis

Our objective was to design, build and evaluate a Learning by Observation system capable of modelling the behaviour of expert agents utilising state representations expressed using various types of Qualitative Spatial Relations. Evaluated in canonical, spatially-situated learning tasks, the system is designed to not only learn but *mimic* the behaviour of expert agents – both human and artificial – and achieve a similar level of task performance. A crucial capability of the system is its ability to alter the granularity of its own representation in order to capture information required to accurately predict particular actions, while at the same time ensuring the compressive benefits of abstraction are not lost. In the pursuit of our goal we encountered many sub-problems relating to the characteristics of observation data, data expressed in terms of QSRs, the selection of learning algorithms and the deployment of learned models on situ-

ated agents. Our approaches to these problems comprise a loose collection of insights into data management that can be generally applied, especially when handling data expressed in terms of QSRs.

This thesis makes the following contributions.

- A full, integrated system for Learning by Observation using Qualitative Spatial Representations, and deploying learned behaviour models as controllers for virtual agents. Experimental evaluation on canonical, benchmark tasks in RoboCup Soccer and Starcraft, two significantly different application domains.

- An extension of the experimental work to include a study of learning from *Human* controllers, as well as synthetic controllers, in the application domains.

- Experimental comparisons between metric and qualitative representations across different task configurations, using a variety of learning mechanisms. Analysis of the characteristics of QSR-transformed data, the problems it poses for learning systems, and practical solutions to those problems.

- Studies of situated agents making use of behaviour models learned from observing both synthetic and human experts. Study of the relationship between classifier accuracy and practical, in-situ performance of agents.

- An application of our system used in conjunction with a Reinforcement Learning approach, where the behaviour of expert agents is used to provide a head-start for the RL process.

# Background

This chapter provides an introduction to the problem of learning by observation, an overview of the recent state-of-the-art in the field, as well as an introduction to Qualitative Spatial Representations (QSRs), and the intersection between these fields. We then survey related work in these fields in our chosen application domains.

## 2.1 Learning By Observation

Learning by Observation is a type of Artificial Intelligence learning problem whereby an agent learns a certain type of behaviour by observing expert agents engaging in that behaviour. This kind of learning can be found in various forms within AI research, with a wide range of definitions of *learning*, *observation* and *expert* being used. Learning by Observation systems have been studied in relation to games such as Tetris [26], with one of the earliest pieces of work being on learning to play Chess from records of human games [26]. One form of the problem is known as *Learning By Demonstration* (or Learning *From* Demonstration), sometimes seen in Robotics [9, 7] where human guidance is used to teach a Robotic artefact, such as an assembly arm to perform particular tasks. Typically here the learning agent is observed by some external means, such as a video camera, or directly by the human expert, and manipulated through a control interface such as a joystick, with these control inputs being recorded. In practice the Robot system then replays what the human has taught it. In this case, the *observation* comes from the Robot system observing the control inputs provided to it by the human expert. In some cases these may be additionally combined with sensor inputs on the Robot, so as to create a sensor-motor mapping. Most LbO work operates under the assumption that experts may not

demonstrate the *entire* range of actions possible in a particular domain – the set of demonstrated state-action pairs will, in any sufficiently complex domain, be significantly less than the set of *possible* state-action pairs – and so techniques to generalise what has been learned can be important. How this is accomplished is highly dependent on the kind of learning system used, as well as the entire system pipeline, and how well it accommodates generalisation. For instance, Floyd [28] uses case-based reasoning, and has the option of selecting the action associated with the state that is *most similar* to any newly encountered state for which there is no entry in the case-base learned from the expert.

Where in the previous example the human expert observes the Robot system externally, LbO problems have been tackled in ways that allow the learner to receive the same sensor input as the expert system, rather than observing it externally [28]. These take the form of a complementary piece of software running alongside the expert's control system, observing all inputs and outputs and producing a resulting mapping. Such systems have been used to teach obstacle avoidance and navigation behaviours [74, 7]. In the cases where sensor input is not shared, it may not be clear based on the form of *knowledge representation* available to the observing agent, which inputs the expert utilises in order to guide its behaviour – this is then exasperated by noisy observations and domain dynamics. Inputs may even be hidden – an agent may have internal state, or behave according to some unknown probability distribution. Problems with these characteristics lie at the more difficult end of the spectrum of LbO problems. In much LbO work, the expert is instead considered to be an oracle which can be consulted at will in case the learner finds itself in a situation for which it has not been trained. The expert is often considered to be ever-present, in a similar way to an objective function in Reinforcement Learning [73], and ready to guide the learner when requested. For instance, in situations using case-based learning where a new case that is not in the existing case-base is encountered, and the expert must be consulted as to what course of action to take. These approaches are often categorised as *active learning* approaches, where the underlying assumption is that the expert is still available for further demonstration, and can interact freely with the learner. This may not always be the case for a variety of reasons, and relaxing these assumptions is one more way to

increase the difficulty of LbO problems. Categories of LbO where the expert is observed by an external learner, with no interaction, are called *passive learning* techniques. It is this kind of LbO that we are interested in, and under the assumption that expert data is finite, with no interaction possible between learner and expert other than observation. An additional distinction can be made between types of LbO by the *intent* of the expert agent – in some scenarios, such as in Robotics domains where humans teach Robot limbs to perform tasks, there is an explicit effort on the part of the expert to *teach* the learner, and so the expert acts in ways conducive to teaching – perhaps performing at a slower speed, and engaging in simpler movements than it would ordinarily. Alternatively in scenarios such as the work of Munoz et. al [58] where the learner learns to control a virtual racing car, the expert is simply engaging in the task naturally as it would outside of a teaching environment, with no alteration of its behaviour. In our work, this is the kind of passive learning we are interested in.

### 2.1.1 Formalism

The work of Montana and Gonzalez describes a small formalism with which to define and further understand the general LbO problem [55]. We make use of this formalism and its language throughout this work, and so briefly introduce it here. To begin, let $B_C$ be the *behaviour* of an expert agent $C$, and let $E$ be an environment in which the agent is situated. . Regardless of origin, during operation, this behaviour is expressed as a Behaviour Trace $BT$ representing the actions the expert performs over a period of operation. The behaviour trace of an expert is then:

$$BT_C = [(t_1, x_1), ..., (t_n, x_n)]$$

where $t$ is a time index, and $x$ is an action. The change in the environment $E$ over time is represented by an input trace:

$$IT = [(t_1, y_1), ..., (t_n, y_n)]$$

The combination of a behaviour trace and an input trace produces a Learning Trace $LT$:

$$LT = [(t_1, x_1, y_1), ...(t_n, x_n, y_n)]$$

This is simply a concatenation of a time index, the state of the environment, and the action the expert agent performs in that state. The Learning by Observation task is to learn a *new* behaviour which approximates as closely as possible the way the expert agents in the environment behave, given these same inputs $LT_1, ..., LT_n$.

In practice, additional mechanisms may be required to be engineered in order to allow the new behaviour to be expressed in a situated agent, given the same inputs and environment as the expert agent. But the main contribution of this framework is to provide us the right language to talk about LbO problems. We found that the formalism sufficiently describes all of the LbO problems in Argall's 2009 survey work "A survey of robot learning from demonstration" [7], with each variant of the LbO problem fitting neatly in to the Montana and Gonzalez framework in one way or another. This shows that in general, this formalism applies to many scenarios, e.g. a sports robot that learns the behaviour of opponents by watching replays of past games, or a disaster relief robot in an ad-hoc team which requires training in the field. Learning by observation can be extremely challenging, particularly if the behaviour of a given expert agent is non-deterministic, or if an agent has internal state which is unobservable by an external entity. A key challenge of the LbO problem is to find the correct *knowledge delivery* in which observations are delivered. This can take many different forms, earlier we described LbO scenarios in which an expert is externally observed by a learner, and also where the expert and learner co-exist and receive the same sensor input. These are different types of approaches to knowledge delivery, and each may impart different perspectives and challenges on learning an identical task. We are interested specifically in *knowledge representation*, which similarly imparts different perspectives on the terms a system uses to understand the world. Davis, et. al [22] give several examples of the properties of a knowledge representation, including *"[...] It is a set of ontological commitments, that is, an answer to the question, In what terms should I*

*think about the world?"*. We think that it is key to find the right terms with which to understand an expert and the world it inhabits in order to learn its behaviour. Along these lines, the work of Akgun et. al [2] addresses the LbO problem through using a novel temporal representation on an iCub Robot learning motor skills such as scooping, pouring and placing. Instead of a continuous time series of states to represent learning traces, a sparse set of keyframes is used instead, and so the Robot learns the behaviour through calculation of the intermediate motor steps between demonstrated keyframes (called splining).

### 2.1.2 Tiers of learning

In addition to the formalism described previously, the work of Montana and Gonzalez [55] identifies four different kinds of LbO, each with an increasing level of difficulty. Summarised as follows.

**1. Strict Imitation** The simplest case, wherein the learned tasks and the environment are purely functions of time. Robots on a manufacturing line are one example, requiring no generalisation.

**2. Reactive Skills** The task being to learn a mapping from perceived environmental states to actions. AI Systems that learn to play video games such as PacMan or Pong are one example.

**3. Tactical Behaviour In Known Environments** Requiring generalisation and planning, considering possible future states. Stategic tasks incorporating elements of hidden information are an example.

**4. Tactical Behaviour In Unknown Environments** The most difficult case, requiring the same skills as 3, however in which the environment and the way it changes is previously unknown and must be discovered.

While this is by no means an exhaustive list of all types of learning by observation tasks, it serves as a useful starting point. This work, given the complexity of the agents and domains in which we will work in later chapters, sits somewhere between **2** and **3**.

### 2.1.3 The Learning by Observation Pipeline

The work of Floyd [27] identifies four sub-tasks associated with building a learning by observation system in the form of a data processing pipeline. This pipeline is the set of practical steps typically required to produce the new behaviour model $B$ in the Montana and Gonzalez formalism, and is found in many LbO systems.

**1. Modeling** Concerning the initial setup of the observing agent – determining how information about the environment and the entities within it will be provided to the agent. This involves receiving raw observational data, perhaps from the output of a simulator or piece of hardware, and potentially transforming it into a certain type of *knowledge representation*. In the formalism, this relates to determining the formats in which $X$ and $Y$, the action and state representations are delivered as part of a learning trace.

**2. Observation** At this stage the agent observes an expert engaging in some task, the observations of which are provided in the format described in the modeling stage. This will be equivalent to a learning trace, or a set of learning traces, as defined in the formalism.

**3. Pre-Processing** Here, an agent may engage in information extraction to determine what parts of the observed input are relevant, and how to map those inputs to the desired outputs. This prepares the agent for model building, and so can encompass feature selection and generalisation phases. In the formalism this is equivalent to a section of the construction of $B$.

**4. Deployment** This phase involves learning, using the potentially transformed data from the previous phase to build a predictive model, which is then installed on a situated agent and used as a controller. In reference to the formalism, this is equivalent to constructing $B$, as well as any practical, domain-specific machinery required for executing the learned behaviour on a situated agent.

By no means are these steps *fixed* however, and some LbO systems may skip one or the other, such as the racing-car driving system of [58] which does not alter the world model provided to

the system. These are just several highly common steps typically found in LbO systems. In this work we make use of the full pipeline. As mentioned, we are interested in the form of *knowledge representation* available to a LbO system, how that representation can be exploited by machine learning, and the *mechanisms of deployment* to best make use of those models. In practice, these are not at all isolated steps, but are interconnected in various ways. The character of LbO systems is therefore that of *integrated systems* with a wide range of possible design choices, and multiple levels of problems to be solved. As such, any one of the above areas could easily be isolated and focused upon in order to produce a research thesis.

### 2.1.4 Research Problems For Learning By Observation Systems

LbO tasks typically lack access to a global, task-level performance measure, and lack access to an observed agent's internal state. Such systems are also often subject to the *limited availability* of training data, meaning that pre-processing steps as in the Floyd [27] framework become particularly important in order to extract as much information from the limited data as possible, and further validate our particular interest in *knowledge representation* in order to extract the most information from what has been observed.

An expert agent is one that has some degree of competence in the tasks associated with a particular environment, and a learning trace is a record of that agent's activity. While the expert agent may itself have access to some global objective function to guide its behaviour, the performance measure of the LbO system will be in terms of how well the learning agent can mimic the expert. The system will not have access to an objective performance measure that can be consulted at will an unlimited number of times, because the system learns only from a *limited* amount of training data provided to it. It may be the case that an agent learns to perfectly mimic an expert that has a low degree of competence. In which case, the observing agent will behave optimally in terms of the LbO task, but sub-optimally in terms of the specific task. The sub-problem of *expert selection* will always be present in such problems, but without some mechanism to evaluate the quality of particular experts it is impossible to solve, and is typically pre-empted by a sparse number of experts being available for a given domain. A

learning system must therefore learn from what is available.

While reinforcement learning systems are capable of discovering new, novel solutions to the probem, they are often prone to overfitting, and such systems might take a long time to train, and again require the existence of a suitable objective function. LbO can in fact be used to address these problems within the reinforcement learning framework [45], and a way of doing so will be described in this thesis. While LbO can be used as a framework to produce predictive models in and of itself, it may also be utilised in conjunction with reinforcement learning by providing an initial model of state-action pairs (learned from an expert) to reduce the initial amount of exploration the learning agent has to engage in. So the frameworks need not be at odds, and can be complementary.

One further problem for a LbO system is that of the observation of internal state. The only inputs to a learning by observation system are how the expert agents *act* and how the environment *changes*. Any internal state of the experts cannot be observed directly, though it may be inferred through observations of external factors, especially if an agent carries out predictable *sequences* of actions – such as a reactive plan. Given knowledge about the action taken at $t-1$, it becomes possible to predict the action at $t$ and $t+1$. This is made easier in cases where actions modify the environment or the observable state of the expert, and is a matter of learning given the right information.

## 2.2 Qualitative Spatial Relations

During our review of LbO work we noted little attention applied to the question of representation, and how different representations can be used in relation to LbO. Much of the LbO work within our area of interest (Agent Control) such as that of Floyd [27] (which we will discuss in more detail later) and [4], as well as LbO systems that play Tetris [70], Chess [26], or drive Race cars [58] keep the representation of the world to the learning system the same as what ever representation the domain outputs. A small amount of work engineers features in to the representation to include certain pieces of information that are not present, for instance in the

work of Konur [46] in RoboCup Soccer, the representation includes variables such as *MyDistanceToOurGoal* and *MyCurrentPlayingRegion* that encode domain-specific information. This is not to claim that those systems are not successful within their given tasks – the experimental and theoretical results of the authors show otherwise. But most work on LbO systems we encountered focussed on the selection and optimisation of learning algorithms, as well as preprocessing steps. Case-based reasoning is a highly popular technique within the literature, so this is to be understood. In terms of the pipeline from subsection 2.1.3 we found work considering the first step (modeling) to be typically under-represented in the literature.

We view *knowledge representation* as being a core issue with regard to the LbO problem. We are specifically interested in Qualitative Spatial-Relational Representations as a form of knowledge representation, and this section justifies that interest. Input to AI systems – from watching video, processing simulated data, raw sensor-motor data, or from mobile robots observing the physical world is typically highly detailed and information-rich, but ultimately noisy and unstructured. The field of Qualitative Spatial Representation and Reasoning [21] is a form of Knowledge Representation that seeks to represent and reason about space and entities within space in a qualitative, rather than quantitative way, addressing several of these problems. We are particularly interested in representations that are not only qualitative, but also *relational*, thus encoding qualitative *relationships* between entities. QR techniques may therefore reveal additional structure in data that was previously inaccessible, without encoding any specific domain information. In the language of Davis et. al [22] QSRs provide a set of terms through which an AI system can understand the world. Since the domains we are interested in are physically-analogous, we are therefore interested in qualitative representation of physically analogous spaces – position, orientation, dimension, velocity etc.

The earliest kind of QR representation is given by Allen's interval algebra [5], first published in 1983. Allen considered temporal events to be specified as a point in time, with an associated duration. Pairwise, binary relations of the form $BEFORE(a, b)$, $OVERLAPS(a, b)$ and $DURING(a, b)$ are then defined. This allows for a robust form of reasoning about the ordering of events, and an explicit representation of the *relationships* between events that was not pre-

| Relation | Representation | Definition |
|---|---|---|
| Equal | $EQ(a, b)$ | $a$ and $b$ share all points in common. |
| Discrete | $DR(a, b)$ | No points are shared between $a$ and $b$. |
| Partial Overlap | $PO(a, b)$ | Only some points are shared between $a$ and $b$. |
| Proper Part | $PP(a, b)$ | all points in $a$ are entirely enclosed within $b$. |
| Proper Part Converse | $PP^{-}1(a, b)$ | all points in $b$ are entirely enclosed by $a$. |

Figure 2.1: RCC5 Relation Table

viously available. Imposing the interval algebra structure on the underlying data then has two particularly interesting effects:

- The *compression* of ranges of continuous values into discrete symbols.
- The *expression* of relational information.

The space that Allen's intervals abstract over is the one-dimensional time line, but the same approach forms the basis of applications to multi-dimensional space in much the same way.

One such approach to qualitative spatial representation and reasoning is based on considering the relationships between *regions of space*, where a region is defined as a , though in some cases the region itself is used as the primitive, this is not the case with our application. Region-based approaches, such as the Region Connection Calculus (RCC) [66] provide a set of binary relations based on the *degree of connectedness* between spatial regions, commonly based on points the regions have in common. There exist different variants of RCC, allowing for different types of spatial information, such as tangential relations, or relations in three-dimensional space, or that deal with polygonal shapes [20]. In our work, we are interested in the RCC5 formulation, which provides the relations in Figure 2.1 between 2D regions. Since we deal with scenarios that are physically analogous, the definition of a region is naturally based on an observed entity's dimensions, and so can be represented as a minimum bounding rectangle [17].

Qualitative representation of other types of space have also been achieved. The Qualitative Trajectory Calculus (QTC) provides a qualitative representation of relative motion [87]. The QTC$_B$ (Basic) variant describes whether two points, *k* and *l*, are moving towards or away each other, to the left or to the right of each other, or faster or slower than each other based on their positions and velocities. The QTC calculus itself has several variants – QTC$_N$ allows for

reasoning about the movement of points in a network, and $\text{QTC}_S$ allows for reasoning about the *shapes* of trajectories of moving points. In our work, we employ the $\text{QTC}_B$ (Basic) relation set, which encodes the following relations between two points *k* and *l*, with each relation taking on a value in the domain $\{$-,+,0$\}$ describing its state. The full $\text{QTC}_B$ relations are as formally specified as follows

**Towards/Away relation**

- - : *k* is moving away from *l*
- + : *k* is moving towards *l*
- 0 : *k* is stable with respect to *l*

**Left/Right relation**

- - : *k* is moving to the left of *l*
- + : *k* is moving to the right of *l*
- 0 : *k* is stable with respect to *l*

**Relative Motion relation**

- - : *k* is moving faster than *l*
- + : *k* is moving slower than *l*
- 0 : *k* is stable with respect to *l*

By default this uses the $+, -, 0$ representation, however this can easily be decomposed into more easily-managable logical statements such as $isMovingFasterThan(A, B)$ where $A$ and $B$ are agents.

The Star Calculus provides binary relations for describing the qualitative direction between points in space with respect to one-another [67]. The Star Calculus employs angular zoning based on either the parameter $m$ (yielding zones of size $360/m$ as shown in Figure 2.2), or through the specification of an arbitrary sequence of angles. The result is a set of circle sectors

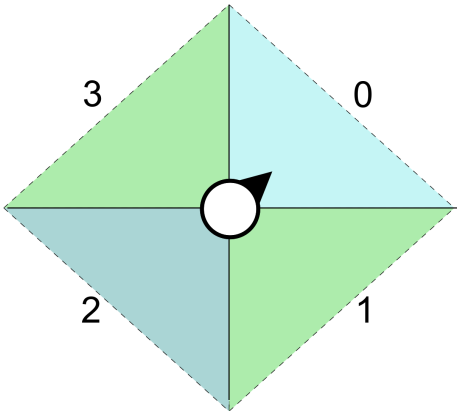


Figure 2.2: Illustration of a Star Calculus representation, featuring four regular, angular zones around a central entity. A second entity inhabiting one of these zones could be said to have a $0^{th}$, $1^{th}$ etc. relation with the centre entity in terms of Star.

emanating from a point, extending into infinity, discretising over a range of angles, with each composing a single binary relation. Between two points the current SC relation is determined by taking the angle between them and determining which discrete zone the result falls in to.

### 2.2.1 Qualitative Distance

We employ the measure of qualitative distance proposed by [18] whereby the distance between two objects $A$ and $B$ is expressed in terms of the presence of $B$ in one of a set of distinct, non-overlapping distance intervals relative to $A$. We centre uniformly-sized intervals at $A$, yielding doughnut-like spatial regions, as illustrated in Figure 2.3.

## 2.3 Application Domains

In our work, we are interested in LbO in adversarial, spatially-situated domains. An adversarial domain is one in which agents compete against an opponent, or opponents, in some task, and possess the ability to act in such a way that disrupts other agents, creating the potential for conflict [89]. We work with two domains that have these characteristics – RoboCup Soccer, and the Real-Time Strategy game Starcraft. The following two sections provide very basic introductions to these domains, before we review LbO work that has been accomplished in them. More in-depth descriptions of the domains are provided in the later chapters, chapter 4

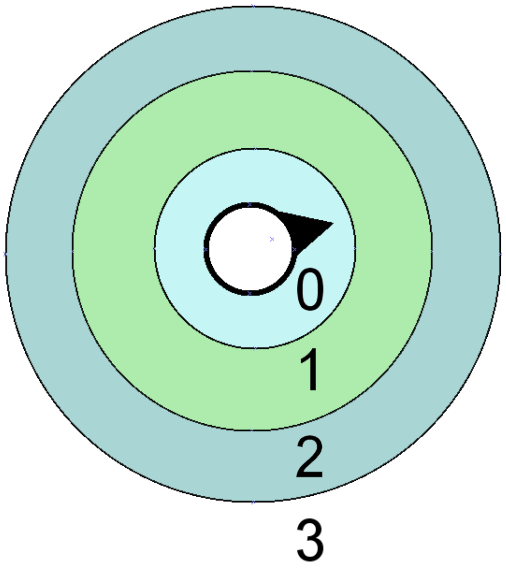


Figure 2.3: Illustration of a Ring-based representation of Qualitative Distance. A second entity inhabiting one of these zones could be said to have a $0^{th}$, $1^{th}$ etc. relation with the centre entity in terms of the distance relation

and chapter 5.

### 2.3.1 RoboCup Soccer

The RoboCup 2D domain is a widely-used multi-agent testbed which simulates games of soccer played by individual playing agents (players) in 2D. It is a widely popular and standardised platform for multi-agent systems research, with competitions and dedicated conferences occurring. We are interested in two sub-problems of RoboCup Soccer – first, the Keepaway sub-game in which a team of *keepers* attempts to maintain control of the ball from a team of *takers*, as well as the full-game scenario which is a simulation of a full 11-a-side game.

At any point in time, all players have a 2D pose (i.e. a 2D position and orientation) and the ball has a 2D position. Given limited sensing of the surrounding players and the ball, on each frame each player selects an *action*: *kick*, *dash*, *turn*, *tackle* or *catch*, which may themselves have continuous parameters (such as the power of a kick, or the degree of a turn). The simulation provides access to the metric positions of all game elements (including static features such as the position of the goals) at every frame, plus the actions the agents took. We refer to this data as the game's *state*, and we can partition this as necessary to maintain the idea that a player only

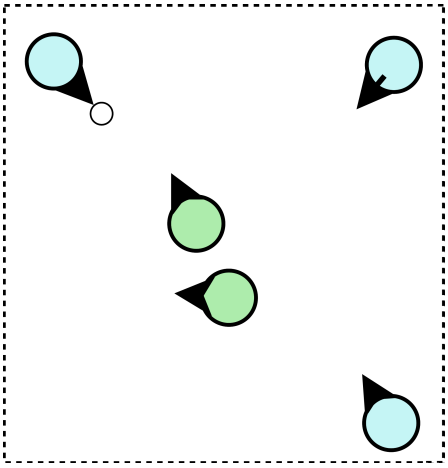

Figure 2.4: A 3vs2 Keepaway Scenario

partially observes the field.

Figure 2.4 shows a typical 3v2 keepaway scenario. The *keeper* team is situated around the borders of the play area, with the opposing team in the centre. The game is split into temporal slices called *episodes*, where a single episode continues so long as the keeper team controls the ball. As soon as control is lost to the taker team (or the ball leaves the keepaway area), a new episode starts.

### 2.3.2 Starcraft

Starcraft is a Real-Time Strategy game released by Blizzard Entertainment in 1998[1]. RTS games typically involve two or more players initially situated in different locations on a two-dimensional landscape, which may have a variety of geographical features such as hills, ramps, valleys, as well as navigable and unnavigable areas. Players start with a selection of rudimentary structures and units, which must be used to gather nearby resources, which are finite. These resources are used to produce new units, building and upgrades, with the aim of producing military units to destroy the opponent's base and win the game. This is accomplished by selecting build options from various buttons and menus.

[1] http://us.blizzard.com/en-us/games/sc/


Figure 2.5: A combat scenario in Starcraft

Military units come in a wide variety of types, differing in speed, armor, movement type (land, or air), thus bearing their own strengths and weaknesses, and are typically segregated according to a player's faction (or race), resulting in an asymmetric game if each player picks a different faction. RTS games are also played under the principle of *partial observability* as each player can only observe the area of the environment directly surrounding their own units and buildings, with the unobservable area being covered by what is called the *fog of war*. In the AI community, each year the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment hosts the Starcraft AI Tournament, which pits Starcraft AI systems designed by teams of AI researchers, or individuals, against each other in a tournament very similar to the annual RoboCup tournament [15]. There are now several similar tournaments occurring worldwide each year, with an ever-growing roster of entrants. It is Starcraft's position as the *gold standard* of RTS games played at the highest level of human competition that makes it an attractive and interesting platform for AI research.

Units are controlled in Starcraft by selecting either a single unit or a group, and clicking a point in space to move to, or an enemy unit for them to attack. Units may also have special abilities that are deployed in a similar way after being selected. Starcraft features three distinct playable races – the Terrans, the Zerg and the Protoss – each having a dozen or so different units available to them ranging from land units, air units and support vehicles.

## 2.4 Existing Work

In the following sections we survey the existing work on learning by observation in the RoboCup and Starcraft domains, and review existing work on learning with QSRs and learning by observation with QSRs.

## 2.5 Learning By Observation in RoboCup

The state-of-the-art in LbO in the RoboCup domain is the work of Floyd [27], whose pipeline we previously adopted in subsection 2.1.3. Here, a case-based reasoning approach is used to harness a database of learning traces. Pre-processing is applied to extract features from this database that provide the most predictive power about an observed agent's next action, allowing the system to weight different parts of the state representation, and to adapt the observed cases to unseen situations by identifying the underlying commonalities in its perceived state. This is achieved using a weighted k-nearest neighbour algorithm [29]. In this work, the approach to generalising the observed learning traces is to build the feature weightings, in order to determine the most important inputs for the predictive model, and factor out features that may be present due to noise or coincidence. Our approach to generalisation is to explicitly represent the environment in qualitative terms, and to later use a *learning mechanism* that can sort out which symbols are the most important. As we use an identical framework to Floyd, a comparison between this approach and our own can be shown visually in Figure 2.6. We argue that our approach has benefits in terms of speed – pre-processing a case base is an offline endeavour, may require much processing power, and elements of knowledge engineering in and of itself to

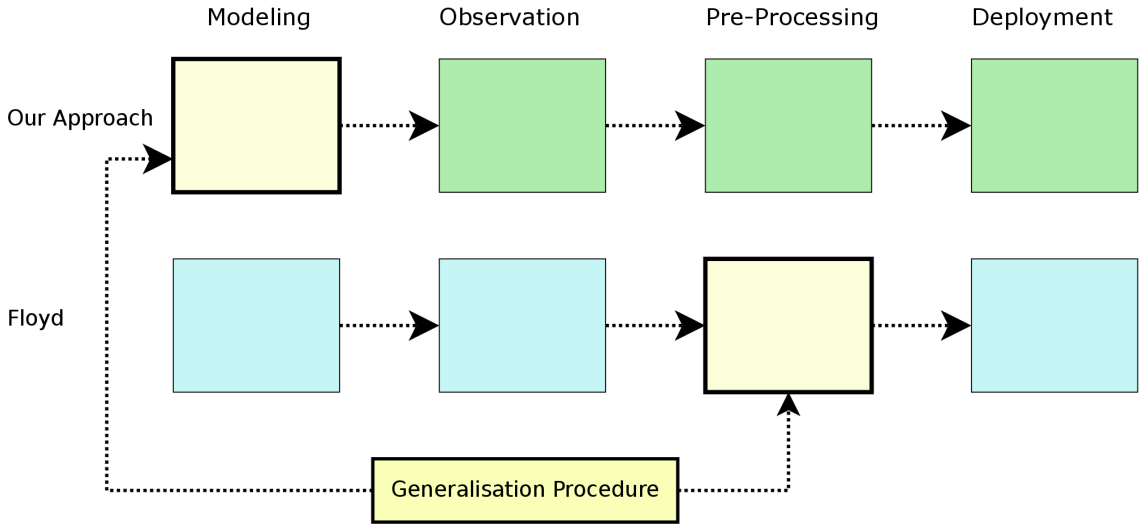


Figure 2.6: The location of the generalisation approach in our work vs. in the work of Floyd [27].

get the best results.

Learning tasks in RoboCup have also been addressed using traditional reinforcement learning, which we now briefly discuss, as it turns out that the state representations used by such systems may be of interest to us, even though the learning approach is not. Vieira [88] employs Temporal Difference learning with the SARSA algorithm [71] to learn an adversarial dribbling task where an agent must keep control of the ball from an opponent. Here, the state-space is defined in terms of 20 continuous variables, including those typically available to a RoboCup agent by default, and several provided through knowledge engineering by a human expert. The selects an action, from a pre-defined set of non-atomic, macro actions. At the end of an episode, an agent is assigned a reward based on its success.

The work of [40] follows a similar approach to the problem of learning a policy for pass selection, where a player in the midfielder role must determine the best team-mate to pass to. Stone [83] utilises reinforcement learning techniques to allow agents to learn to play the RoboCup Keepaway task described previously, learning co-operative team behaviours using Neural Networks. This work, and the similar work of Jung [43] are typical of reinforcement learning approaches which assume task-level reward functions, and often the ability to engage in an unlimited number of trials [44].

```
MyPlayerType = 2:
| BallDistanceToOpponentGoal <= 18.2609 :
| | MyCurrentPlayingRegion = 4:  9 (6.0/1.2)
| | MyCurrentPlayingRegion = 5:
| | | MyDistanceToOpponentGoal <= 12.4953 :
| | | | MyDistanceToOurGoal <= 92.6233 :  1 (3.0/2.1)
| | | | MyDistanceToOurGoal
```

Figure 2.7: Decision tree fragment from [46]

Recent work [94] shows how probabalistic models can be built of agents engaged in a penalty shoot-out scenario, which exists as one of the sub-problems of RoboCup soccer. The decision process of agents is learned, which is characteristic of the reactive skills described in subsection 2.1.2. The representation used – optical flow vectors – is relatively inflexible, and does not itself extract robust, re-usable models due to the representation being tied to pixel-level data.

In [78] reinforcement learning is used to construct agents that learn co-operative behaviour with teammates that take fixed control policies, again optimising against an objective function. The state representation is once more based on continuous variables, and engineered knowledge, including some relational variables.

In [46], a player learns a dribbling task by building a decision tree based on guidance from an expert agent. The state representation includes 35 hand-coded variables, many with continuous state spaces, that describe the world. The use of decision trees allows the learned model to rank features in the representation to locate those that are most salient to a particular action. The previously discussed work of Floyd [27] accomplishes this by extensive pre-processing of the training data,to determine feature frequencies which are then used as weights in the prediction task. Here, this step is part of the classifier, which is the C4.5 decision tree classifier [63].

It is interesting to note that in the existing work, when knowledge engineering is used to add information that is not immediately present in the default state representation provided by RoboCup, it is often in the form of *relational* information. For instance, in [78] one state variable given to the agent is the continuous variable *"Angle between the opponent goal and the ball"*, and the twelve additional variables added to the state space of [88] are almost entirely

relational in nature, including such things as *"Angle of the opponent in relation to the agent's body"*. In [46] highly specific state variables are used, such as *MyDistanceToSecondVisibleOpponent* and *MyAngleToFirstVisibleTeammate*. In our work, since we represent the state of the world as a set of spatial relations, we capture such information by default without encoding it ourselves in such domain-specific terms.

## 2.6 Learning By Observation in Starcraft

The majority of learning by observation work in Starcraft centres on the learning high-level, goal-based strategies (macro-management, as defined previously). As is characteristic of the Real-Time Strategy domain, in Starcraft players gather and spend resources on various different buildings and combat units – each of which may have their own unique characteristics in terms of size, movement speed, damage, special abilities etc. Some structures may have a defensive function, while others may provide access to certain upgrades or unlock the production of certain units. The sequence of buildings a player elects to buy forms the basis of their high-level strategy, allowing access to certain combat units which can then be leveraged against the opponent. This sequence is known as a *build order*, and a player with knowledge of their opponent's build order may alter their own build order in order to produce units that provide a counter to those units the opponent will field. As might be expected, this is not entirely straightforward, as the game is only partially observable to each player, only allowing a player to observe areas of the map that are inhabited by their own units. Determining an opponent's build order requires scouting and the ability to infer which units an opponent will have access to, based on which buildings are present.

Much work has looked at learning build orders from replays of expert players, as well as learning them from scratch using reinforcement learning [95]. In [84] a Bayesian approach is employed to build a probabilistic model used to make predictions about the build order a player will follow, given observed evidence. A variety of different learning mechanisms have been applied to the same task – in [24] Hidden Markov Models [64] are used, while [68] uses

Case-Based Reasoning, and [31] uses Neural Networks.

Our work however focuses on a problem that has received less attention in the literature – the problem of how an AI system manipulates units in *combat scenarios* in order to overcome an opponent. There is a lot less work on this kind of task. One reason might be that the micro-management mimicry task is potentially harder than the macro-management mimicry task, as it deals with a continuous state-space in terms of unit positions and commands. Combined with the natural variation in the game, this can result in a large number of highly varied scenarios from which to learn.

In recent work that does consider the micro-management problem [60], it is tackled using a Bayesian network using data gathered from observing human operators in small combat scenarios. The model is used to produce agents to control units that seamlessly and autonomously co-operate with a human operator, complementing the playing style and tactics of the human. The state-space for the model is partly based on metric information, such as the raw distances between units, as well as some hand-coded symbols. We will survey this work in greater technical detail in later sections, as we repeat the same experiments using our own system, providing a direct comparison.

## 2.7 Learning with QSRs

Much work has been done in regards to utilising QSR techniques in learning and classifying observed human activities. Typically such approaches have dealt with scenarios where there exist noisy observations, such as in the processing of video, and feature interactions between multiple entities. Our work fits into the same broad framework of QSR-based representations coupled with machine learning used by much of the existing work. The work of Sridhar [79] is seminal work that uses QSRs to learn classes of spatio-temporal events from video data, using RCC5 and Allen's temporal relations. Bounding boxes around observed entities provide the basis for spatial regions, and used to generate QSR representations of metric observations from fixed-viewpoint cameras. The *interactions* between spatial regions over time are used to

discover classes of events in an aircraft loading domain in an unsupervised way, and then predict the probability of a given event when observing previously unseen interactions.

In [56], activity recognition is accomplished in a basketball domain, though instead of a representation based on the relations between spatial primitives, as supplied by RCC in [79], the state space is given as an ad-hoc collection of domain-specific qualitative relations that blend spatial information with knowledge about the basketball domain. The predictive model then takes the form of a Markov Logic Network, a model which has been previously applied to such action recognition tasks with success [13], and is used for event annotation.

A similar approach is used in [10] for monitoring of industrial activities. Interestingly, here rather than a static, fixed-viewpoint camera pointed at a scene, traces of behaviour are gathered from cameras and on-body sensors mounted on an operator, providing an egocentric representation. The qualitative spatial relations between items the operator manipulates and their body are then used as the state representation in models of activity.

Much of the existing work on using learning and QSRs focuses on *recognition* and *annotation*, typically in unsupervised scenarios. Whereas we focus on supervised learnings tasks that also have a control component. While the control component is not itself novel – QSR based representations have been used to program autonomous agents such as mobile robots [80] – there is far less work on combining QSRs with learning and control systems in an integrated way.

## 2.8 Representation Learning

One characteristic of QSRs is that they often possess certain parameters that govern the degree of abstraction they provide. For instance in the Star calculus discussed previously in Figure 2.2, the interval of the angular zones can be modified to alter the granularity of the state space. In other words, these parameters govern what *spatial features* are present in the state representation. The field of Representation learning [11, 50] is a field of study that, in part, seeks to analyse how systems can learn ways to represent the world around them. This is quite broad,

and encompasses a wide range of systems, from those that learn foundational representations from raw sensory data [47], to those that modify parameters associated with their own perceptions of the world [50]. The work of [75] outlines a rough framework for Representation Learning:

1. Instantiate a new feature space.
2. Transform the training data into the new feature space.
3. Train a model on the transformed data.
4. Evaluate the effectiveness of the model.

The core observation of this kind of work is that the performance of machine learning algorithms can be heavily dependant on the representation of input data provided to them. The goal of representation learning is to explore the space of possible representations of that input data in order to find those features that can be best leveraged by classifiers [91]. Manual representation engineering of this kind would be laborious and time-consuming, especially given the large space of possible features a representation might have, as well as their possible combinations, and so automated methods are preferred. Much of the current work in the field [75] is concerned with discovering representations for, and inside, deep learning networks, but many of the developed techniques and ideas are general and can be applied to any machine learning technique.

In the work of Coates [19] K-means clustering is used to compress unlabelled training data into clusters before being provided to a learning algorithm, and the work of Mahmood [50] investigates representation learning through search in an on-line learning task, where combinations of pre-specified features are iteratively learned to determine the most useful subsets. We are interested in using similar techniques for learning parametrisations of QSRs, in order to discover and amplify spatial features in the data that might provide strong predictive power when predicting the behaviour of an expert. In the example of the Football player taking a shot at the goal in the introductory chapter, there are certain pieces of information that have may

more value than others when attempting to predict what either player will do next. Specifically this information is the angle of the goal keeper with respect to the goal-post and his opponent. In situations like this we may wish to have a large amount of information about that particular angle, but in other situations it may not be as important. We expect this similar sort of structure to exist in our target domains, where specific spatial features are of more importance to certain actions than others. However, discovering what these might be by hand and encoding them into a system would be laborious, and so we will apply ideas from Representation Learning to learn them. We will discuss the implementation of our Representation Learning system for QSRs in the next chapter.

## 2.9 Chapter Summary

In this chapter we introduced the following concepts:

- The Learning by Observation problem, formalism and pipeline
- Qualitative Spatial Relations
- The adversarial domains of RoboCup Soccer and Starcraft
- Representation Learning

We discussed how Learning by Observation problems may have the characteristic of having access to a *limited* number of examples, and so must utilise methods to extract the most information from those examples as possible [7, 27]. In the existing LbO work the consideration of representation as a way of doing this is however not often given much attention, and where the inclusion of abstract or relation information occurs, it is often *ad-hoc* and *unprincipled* [46, 78, 88]. We looked at some LbO tasks wherein data was pre-processed to extract spatial or task-related information about entities, which was then leveraged to build stronger predictive models later on [27]. We briefly discussed the field of Qualitative Spatial-Relational Representation and Reasoning, and our interest in using Qualitative Spatial Relations as a way

of representing abstract and relational information in LbO systems, from the viewpoint that *knowledge delivery* is an important factor to consider in the LbO task. We then looked at the research area of *Representation Learning* which studies how the parametrisations of state representations for learning systems might be improved through learning in order to improve the performance of predictive models [75]. We then described how the ideas presented by this could be applied to QSRs in order to explore the space of potential QSR representations of a metric state [50].

# Framework Overview

. We will look at the *practical implementation and integration* of the Learning by Observation framework, along with representation learning and model deployment. The chapter provides an explanation following the logical course of data through the system, and describes the algorithms in use at various points in the system to achieve its design goals.

## 3.1 Overview

Our overall architecture follows that of the Learning by Observation pipeline from the previous subsection 2.1.3, and is a specific implementation of this pipeline. The system takes as input observations of the states of agents operating in a particular domain, this data composes the learning trace as defined in the LbO formalism given in subsection 2.1.1. At each discrete time step $t_0 ... t_n$ the metric state of the world is observed, including the positions of any agents and objects that might exist, alongside labelled actions taken by those agents on that time step. Actions may be discrete, or have metric parameters themselves. The learning traces are then fed in to the Representation Learning system, where QSRs are applied to convert metric representations, and find representation configurations that provide the best predictive models for predicting an action based on the input of a particular state. These models are then used as part of a control system that uses the predictions from learning algorithms to control situated agents.

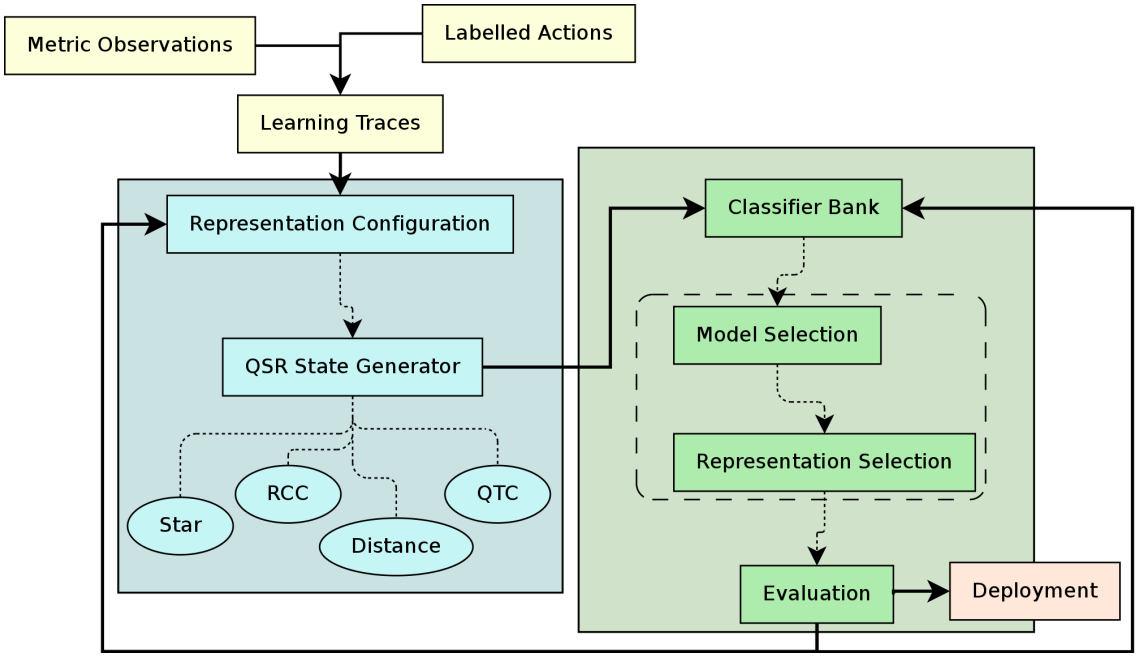


Figure 3.1: System Overview

### 3.1.1 Input

#### States

Recall the definition of a learning trace from subsection 2.1.1:

$$LT = [(t_1, x_1, y_1), ...(t_n, x_n, y_n)]$$

Where a particular $x_i \in X$ represents the state of the world at $t_i$. In this case, $x_i$ is then the metric representation of the world as described above. This is equal to the raw output from a specific domain of interest, such as the RoboCup Soccer simulator or Starcraft. An observation is a time series which records the metric properties of objects and agents engaged in a task in a particular environment.

| $k_1$_pos | $k_2$_pos | $k_3$_pos | $k_4$_pos | $t_1$_pos | $k_1$_action |
|---|---|---|---|---|---|
| -0.0051:0.3521 | -9.7629:-8.9396 | 8.9715:-7.7735 | 8.8845:9.3307 | -9.8372:7.3948 | KICK(100.0) |
| -0.0051:0.3521 | -9.7629:-8.9396 | 8.9715:-7.7735 | 8.8845:9.3307 | -9.8372:7.3948 | TURN(32.1) |
| -0.0051:0.3521 | -9.3127:-8.5426 | 8.3718:-7.6195 | 8.3917:8.9497 | -9.6039:6.8512 | TURN(35.4) |
| -0.0051:0.3521 | -8.7459:-7.9427 | 8.1209:-7.5398 | 8.202:8.8 | -9.3245:6.1308 | DASH(100.0) |
| -0.079:0.9319 | -8.1068:-7.2985 | 8.0284:-7.5076 | 7.8298:8.2189 | -8.9671:5.3012 | DASH(100.0) |

Figure 3.2: Sample output from RoboCup Soccer simulator showing only positional information about the world as observed by the agent $k_1$ along with its actions on the far right column. The columns labelled $k_2, k_3$ and $k_4$ and $t1$ are the observed positions of other agents in the simulation.

This data may take the following forms:

**Positional**

The positions of entities in space. Expressed as the continuous tuple $(x, y)$.

**Orientational**

The orientation of each entity in terms of an angle $\theta$.

**Locomotional**

A vector quantity $V$ of each entity describing the rate of change of its position.

In Figure 3.2 we show a sample five frames of positional output from the RoboCup Soccer simulator observed from the point of view of a single agent. Data may vary dependent on the domain and the task – for instance, RoboCup soccer provides 2D positions, orientations and velocities of players and the ball, however in Starcraft there is no orientation or velocity information present, as this does not feature in the physics model of the game. The facing direction of a unit (equivalent to a player in RoboCup Soccer) in Starcraft is purely cosmetic, and has no functional purpose other than for animation, meaning an agent can move or fire in any direction at any time. Units in Starcraft also move at a constant speed defined by their specific unit type. In Starcraft locomotion is achieved by sending movement commands to move units to specific *points in space*.

**Actions**

Accompanying a metric observation of a state is also a record of the actions agents took in that state. This is again dependent on the domain. Actions in RoboCup continuous parameters – such as the degree of a turn, or the power of a kick, as shown in Figure 3.2. Actions in Starcraft are mixed between those that are entirely discrete with no parameters, and those that have multiple continuous parameters – a move action that specifies a point in 2D space to move to may take the form $MOVE(245, 433)$, specified in a global co-ordinate frame. The action to execute a special ability that has no parameter would just take the form $ABILITY_1()$ and thus be a discrete action, though there are actions that take as parameters points in space, as well as friendly or enemy units. This varies from ability to ability in Starcraft, though due to the example tasks that we selected, we do not largely deal with units that have these capabilities.

### 3.1.2 Processing

The metric states are taken and processed into qualitative states made up of the QSRs described in section 2.2. The qualitative state at a particular time step is then the concatenation of all binary relations that hold between the observable entities in that instant. In RoboCup this may be different for each player, as they are only able to observe the players in their direct field of view. Players are however aware of the existence of all other players playing the game, though may not be able to observe information about all of them. In Starcraft the situation is similar, but the global effect of the Fog Of War means that every unit perceives the same thing, and enemy units outside the immediate vicinity of a friendly unit are unobservable.

The definition of what constitutes a learning trace remains the same, except now each metric world state $x_i \in X$ is replaced with its corresponding qualitative world state. These relations are calculated exhaustively between all pairs of observable agents. In the cae of RoboCup Soccer, to model the restriction on field of view imposed, we discard any relationship involving an entity which is outside of a player's $90^\circ$ view-cone as this should not be used to learn or predict the behaviour of that player.

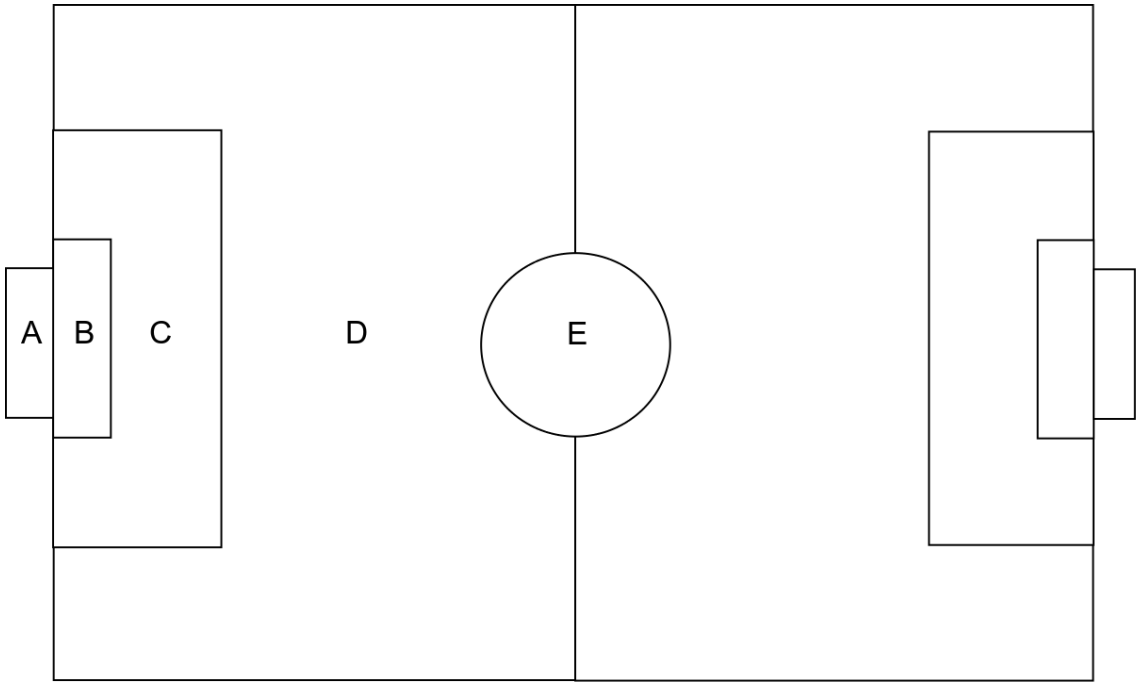


Figure 3.3: Illustration of the regions of the pitch we partition with RCC

### 3.1.3 Qualitative Spatial Relations

In the full game scenario of RoboCup we employ RCC5 to encode a qualitative representation of the game pitch. This partitioning is shown in Figure 3.3 with one side of the different pitch regions labelled. This follows the traditional partitioning of the pitch as according to the rules of the game, but there do exist other areas that are typically referred to, such as the wings and the box, and so other kinds of partitions upon this space are certainly possible.

We make use of the Qualitative Trajectory Calculus for representing the relative motion between entities. Our implementation follows that of [23], using the position, velocity and orientation of entities to generate the relations described in the previous chapter. In addition we make use of the Star calculus, we calculate the pairwise Star relations between all players and the ball. In the case of RoboCup we ensure that these relations are considered in egocentric terms, orienting the $0^{th}$ relation in front of the player's body. Finally we employ the qualitative distance measure [18], again calculated egocentrically with respect to a player.

## 3.2 Learning

In order for our system to learn we must first address the dual problems of selecting and configuring an appropriate learning algorithm, and also of selecting and configuring an appropriate state representation via Representation Learning. As described previously, the QSRs we use are subject to certain parameters, and we are particularly interested in how these parameters can be modified in order to discover useful knowledge about the world. In practice we address these problems concurrently and ultimately address them within the same framework. In the following sections we describe the component parts of the problem before discussing our unified approach to solving it.

### 3.2.1 Selection Problems

#### Classifier Selection

The field of machine learning has developed a variety of learning algorithms that perform generally well off-the-shelf on a wide range of problems, with it often being the case that certain algorithms perform better on particular problems than others [38]. Determining which algorithms will work best on a particular problem is often a case of trial-and-error, and given a set of candidate algorithms, each will typically have a space of possible parameterisations that can alter its performance, often radically. This is solved in machine learning by the approach of hyper-parameter optimisation, where search and optimisation algorithms explore the space of possible configuration parameters.

To illustrate, consider an imaginary algorithm $I$, a performance metric $M \subset \mathbb{R}$ , and two parameters for the algorithm $\phi$ and $\lambda$, which may take particular values such that $I(\phi, \lambda)$ gives some value $m \in M$ on some data input. The parameters for $I$ are:

$$\phi \in \{1...100\}$$

$$\lambda \in \{5, 4, 2\}$$

To optimise $M$ the parameter space $\phi \times \lambda$ must be explored. One common way of accomplishing this is via a grid search [12], exhaustively evaluating each point in the space, to find the best values of $\phi$ and $\lambda$ to use. Due to the exhaustive nature of the task, this can be expensive and time-consuming depending on the algorithm and its parameters. In the case of continuous parameters (as in a Support Vector Machine), a discretisation of the space is applied, the granularity of which may itself be the subject of an optimisation step.

**Representation Selection**

Our system lies in a class of learning systems that have the ability to re-analyse the training data they are provided to extract important features that provide the best classification performance, effectively warping the training data. The QSRs that make up our state representation depend on certain configuration parameters (for instance, the discretisation of angles in the Star calculus) which can be tuned.

We can illustrate this with a simple example. Consider an imaginary agent $x$ which models the decision rule of a proximity alarm sensor: *If motion is detected within 3m, sound an alert*. On each time step, the sensor either engages in a NOOP, or executes an ALARM action based on this rule. Given some labelled observations of the sensor operating, we wish to discover the rule using the QSR ring calculi described in the previous chapter. To do this, we need to find the right parameter $r$ that best discretises space in order to distinguish between states where an agent is within 3m, which will be labelled with an ALARM action, and states where there is no agent within 3m, which will be labelled with a NOOP action.

Doing this by hand would require a human engineer to know (or spend their time discovering via a manual search) which features a good representation needs to have. This may be cumbersome and time consuming when instead we can treat the variables that control the generation of QSRs the same as the hyper-parameters used in learning algorithms, and leave the system to discover the best representation configuration for a particular task by itself. This is in fact desirable, since when observing a previously unseen agent, we do not know *a priori* what the decision rules being used are – we can only observe how the agent acts in terms of its affect

on the environment – so finding representations that provide the right information is crucial.

Finding the best predictive model for us then not only requires addressing the classifier selection problem in order to find a good learning algorithm, but also concurrently addressing the problem of *representation selection* to find the state representations that provide the information required to build good predictive models.

### 3.2.2 Parameter selection with random search

The parameters that control learning algorithms and those that control QSR state generation can be thought of as being equivalent in the task of finding good classifiers. However, hyper-parameter optimisation of learning algorithms alone is a difficult problem even before adding in the extra complexity of our representation configuration. Grid search scales poorly as the dimensionality of the parameter space increases, and so we would ideally prefer a faster and more efficient method. The approach described in the recent work of James Bergstra and Yoshua Bengio [12] provides this, and hinges on randomly sampling the parameter space, rather than exhaustively searching through it. This approach is limited by a fixed number of trials, and so can be as speedy or thorough as required. Though this is still a parameter that is tuned by hand in our present system, we selected $1000$ iterations for the value. Where $A_{ap}$ is the set of hyper-parameters for a given learning algorithm $a$ from the set of algorithms $A$, $Q$ is the set of parameters governing QSR state generation. Each parameter will have a *domain* in $\mathbb{N}$ corresponding to the range of possible values it can take. Finally, we give $m$ as a learning trace of metric observations and the function $qsr(Q, m)$ transforms the metric learning trace into a given QSR representation. This is shown in Algorithm 1.

The only difference in our approach to that of [12] is that we employ an explored set $E$ to guard against re-visiting solutions that have already been evaluated. This makes the algorithm a form of Tabu search [33], rather pure random search. One key observation of Bergstra and Bengio is that not all hyper-parameters are equally important to tune. This principle also extends in our case to the parameters that control QSR state generation. In addition to this search, our training function also looks at the possible value of *previous states*.

**input :** A metric learning trace $m$, a set of learning algorithms $A$ and the set of configuration parameters $Q$ governing QSR state generation.
**output:** The best classifier and representation for the training data.
**begin**
  **foreach** $a \in A$ **do**
    Initialise: $P_a \leftarrow$ The set of hyper-parameters for $A_a$. Initialise: Set $p \in A_{ap}$ and $q \in Q$ to random positions within their domains.
    **while** $outerIterations < oMax$ **do**
      Apply Gaussian Noise to $q \in Q$
      $c = 0$
      E $\leftarrow \{\}$
      $X \leftarrow A_{ap} \cdot Q$
      **while** $innerIterations < iMax$ **do**
        Randomly alter each $x \in X$, ensuring $X \notin E$
        $Y \leftarrow X$
        $S \leftarrow qsr(Q, m)$
        $v \leftarrow train(A_a, A_{ap}, S)$
        $vScore \leftarrow eval(v, S)$
        **if** $vScore > vBest_{score}$ **then**
          $vBest \leftarrow (vScore, a, A_{ap}, S, v, Q)$
          $X \leftarrow Y$
          $E_c \leftarrow X$
          $c{+}{+}$
        **end**
      **end**
    **end**
  **end**
  **return** $vBest$
**end**

**Algorithm 1:** Parameter Search

We evaluate candidates in the classifier-representation pair space with a ten-fold cross-validation, taking the Matthews Correlation Coefficient as a measure of performance [53]. This performance measure is a way of fairly comparing classifiers in situations where one class may dominate another in the training data, which is seen in our data. Apply regulation using the squared $L_2$ norm over the representation complexity, which in our case is the granularity of the QSRs we use, the number of possible states. This form of regularisation is simply a mechanism of applying a penalty for complexity to the performance measure of a classifier [57], meaning the search prefers *conservative* representations.

## 3.3 Output

After this the training process is complete, the system will have a predictive model of the expert agent, given the same inputs, and will have learned a particular configuration of QSR parameters for each action the agent was observed to engage in, as well as a temporal window for each action (which may be between 0 and 5 previous states) describing previous states, expressed using the same QSR representation. The next step is to deploy this model on a situated agent. To do this, we instantiate an agent in a particular domain and begin providing it metric observations of the world. The high-level algorithm for this is shown in Algorithm 2.

**input :** The currently observed metric state at $t$ $m_t$, a trained model $l$
**output:** The action an agent should take in this state.
**begin**
  $S \leftarrow qsr(m_t)$
  $k \leftarrow predict(l, S)$
  **return** $exec(k)$
**end**

**Algorithm 2:** Model Use

The algorithm fragment $S \leftarrow qsr(m_t)$ transforms the current metric state into a QSR state, and attempts to match the current state with a state in the model, by applying the configuration parameters associated with states in the model to alter the perspective of the agent on the currently observed state.

Model use is dependent on the form that actions take in the domain. The simplest scenario

would be one with a discrete action set. Here there is very little more work to be done other than finding a good representation and classifier, as classification accuracy will be closely linked with the performance of the mimicking agent.

### 3.3.1 Forms of Output

Since we approach the general learning task as one of classification, we must deal with the problem presented by actions that may have both discrete and continuous parts, as in RoboCup and Starcraft where discrete actions have continuous parameters. Continuous parameters present a problem for a classification-based approach, as an action which might have parameters that vary will result in a number of unique class labels in the data. This was shown previously in the RoboCup data sample in Figure 3.2, where there are two instances of the $turn$ action. We will discuss how we approach this issue further in the following sections, however in general we have developed approaches to deal with cases where action representations have one of the following characteristics:

- Actions are entirely discrete.

- Actions have both discrete and continuous parts.

- Actions have both discrete and continuous parts, to which QSRs may be applied.

Our approach to predicting continuous action parameters is based around domain-specific *implementation procedures* which function as reactive planners that *calculate* the parameters of an action, given a prediction from the learned model of the discrete part, and the current QSR state.

For each domain we define an implementation procedure for the model (given as the *exec* function in Algorithm 2), and in the following sections we describe the specific implementation procedures for our domains of interest.

| Discrete Action | Parameters | Range | Description |
|---|---|---|---|
| kick | $Power, Angle$ | -100 ... 100 | Accelerate the ball at $Power$ by $Angle$. |
| dash | $Power$ | -100 ... 100 | Accelerates the agent in the direction of its body. |
| turn | $Angle$ | -180 ... 180 | Turns the player's body direction by $Angle$. |
| turn_neck | $Angle$ | -90 ... 90 | Turns the player's field of view by $Angle$. |

Figure 3.4: Example of several actions found in RoboCup Soccer [41]

**In RoboCup**

shows several example RoboCup actions along with their parameters. These appear in the learning traces produced from the RoboCup simulator, as shown before in the sample data of Figure 3.2, where the actions as given above serve as class labels. To predict, a pure classification approach may be very difficult, as the data could potentially contain a large number of classes – equating to one class for each possible value of a parameter for each action (for instance, $dash(-100)$ ... $dash(100)$ could be represented). This initially seems as though it would be a poor approach, however under certain circumstances it may not be. In some cases agents may have *characteristic action profiles* – by this we mean that agents may select actions and parameters from a static, pre-defined set in their code. For instance, an agent may be coded to only ever execute a $KICK$ action with a power parameter of $75$ and an angle parameter of $0$ – in this case, while the parameters are functionally continuous, they remain static and so the concatenation of the discrete and continuous parts can be treated as a single discrete class label. The learning and control tasks are then in their simplest forms, assuming enough training data has been gathered to represent enough of the agent's action space. The implementation function in such a case simply the execution of the action specified by the class label.

The more difficult case, and the more common, is encountered when agents calculate action parameters on-the-fly based on their current observations. Such observations may be noisy, affecting the decision process, and thus the resulting action parameters.

Our approach is to initially disregard the continuous part of the action, and rely on reconstructing it via calculation. Given the current state, and a series of future states in the learn-

ing trace that follow on from it, that we call the *lookahead*, the procedure must find parameters for the action predicted by the behaviour model that will most likely result in those future states, . The lookahead is taken from the training data, and are the states of the world that occurred *after* the expert executed the action the model currently predicts. Since we can never be sure that the same states will follow, it is most accurate to characterise this as the *expected* lookahead, and so we calculate this as the *most likely* set of proceeding states within a finite lookahead window $w$.

**input :** The current state $cur$, the action taken $a$, a window size $w$
**output:** The most probable set of states to follow.
**begin**
  $L = \{\}$
  $c = 0$
  **foreach** $state \in trainingSet$ *where* $action == a \;\&\&\; state == cur$ **do**
    $t \leftarrow state_{time}$
    $seq \leftarrow trainingSet(t_{n+1}...t_{n+1+w})$
    $L_c \leftarrow seq$
    $c{+}{+}$
  **end**
  **return** $most\_common(L)$
**end**

**Algorithm 3:** Finding the most probable lookahead

. Since our goal is to deploy the learned model on a situated agent, we make the assumption that the agent will have access to information about the dynamics of the world in the same way as the expert agent. In RoboCup, this is the likely state of the world in $n$ frames if the ball is kicked with a certain power at a certain angle, and the velocity remains constant for those few frames. In Starcraft this is simplified slightly, since all units move at a constant speed, which may be 0, we can again attempt to predict the future result of an action.

In the work of Floyd [27], the approach taken to action parameter selection is to select the average value of the parameter associated with the action taken in a particular state. This may not reflect the true value of the parameter the observed agent picked in this state, and may not result in the replication of the desired behaviour at all. The approach of Floyd is not entirely without use however, and we regard it as an important first step. In order to find the correct action parameter, we use this value as a first-guess to generate an initial candidate solution. This

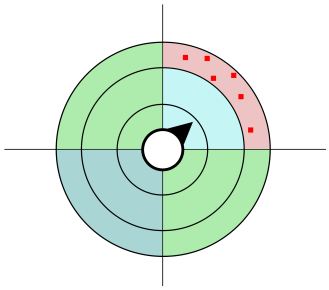

Figure 3.5: Compression of points.

provides a starting point from which to perform a stochastic search over the action parameter space.

### In Starcraft

In Starcraft we simplify much of the problem by applying QSRs *to the action space*, since in a movement command, an agent will move to a specified *point in space*. The set of possible points an agent can be commanded to move to may be large ($4^{10}$ in the worst case). We would prefer not to represent movement actions in terms of absolute global co-ordinates, since this is brittle, especially when we consider differing map dimensions – some points that exist on 128x128 tile map (with potentially $4^9$ possible movement points, in the worst case) may not exist on a 64x64 tile map ($4^8$ potential points). This is particularly prevalent with traces of human players – the lack of pixel-perfect accuracy in selecting movement points means that a human player may select points that are qualitatively identical, but are represented quantitatively as many separate and distinct points. To capture this, rather than representing movement points quantitatively, we represent the parameter of movement actions in terms of the QSRs between the point and the agent commanded to perform the action.

In Figure 3.5, we give a trivial example of how this is accomplished. If we consider the red dots as the relative metric positions of points the agent has been commanded to move to, we can see that (under this particular configuration) they can be expressed in terms of their QSRs with the agent – $(star(0), ring(2))$.

Movement actions are given in terms of QSRs, however actions in the game world are given as metric positions, and so the *implementation procedures* must be one which can translate the

QSR specification in the learned model back in to metric positions in the game world in order to command units. The problem is to find metric points that match the QSRs specified in the action representations. Given a move action specified qualitatively, we wish to locate a point in metric space that best matches that description. As discussed previously, the number of points an agent can move to on a map may be very large, and so calculating the QSRs of every point to find those that best match would be expensive, and consider many superfluous points.

We employ a Monte Carlo sampling-based approach [6]. We randomly generate a set of points around the agent, then calculate the QSRs of these points, and rank them based on their distance from the target QSRs. The notion of distance in QSR space is given by conceptual neighbourhoods which act as transition matrices for QSRs [34], an aggregation of the number of transitions between an origin and target set of QSRs can be employed as a distance measure. A threshold $k$ over this distance measure is used to vary how closely the QSRs of candidate points must match our target QSRs, with $k == 0$ being an exact match. We envision in complex application domains there may exist a trade-off of generating more candidate points $vs.$ relaxing the closeness of matches, and so we leave these as parameters that may be tweaked. In our work with Starcraft, we are able to utilise $k == 0$, and recover . We then define a set of functions for selecting a point $x_i \in X$, these include $closest(X)$, $furthest(X)$, $max\_x(X)$, $max\_y(X)$ among others. For all of our experiments we utilise $closest(X)$ which returns the closest point to the agent that best matches the desired QSRs. Our use of the Region Connection Calculus ensures that points must fall into regions the agent can legally move through.

## 3.4 Chapter Summary

In this chapter we primarily looked at how we achieve a practical implementation of the Learning by Observation Pipeline from the previous chapter. Our approach follows these major steps:

1. The system receives example Learning Traces from a chosen domain, with a metric representation of the world annotated with actions that agents took.

2. The system uses stochastic searches to find, for each action, a state representation in

terms of QSRs that maximises its predictive accuracy over a ten-fold cross-validation. The system may also take into account previous states, if it improves performance.

3. The system then employs various model deployment mechanisms to enable a situated agent to act in the domain using the behaviour models as a guide. Where action parameters must be supplied, it uses domain knowledge to estimate what those parameters should be, based on the states in the training data that proceed the action currently being predicted by the model.

# RoboCup Soccer

## 4.1 Domain Overview

The RoboCup 2D domain is a widely-used multi-agent testbed which simulates games of soccer played by individual playing agents (players) in 2D. At any point in time, all players have a 2D pose (i.e. a 2D position and orientation) and the ball has a 2D position. Given limited and noisy sensing of the surrounding players and the ball, on each frame each player selects an *action*: *kick*, *dash*, *turn*, *tackle* or *catch*, which have continuous parameters (such as the power of a kick, or the degree of a turn). The simulation provides access to the metric positions of all game elements (including static features such as the position of the goals) on each discrete time step, plus the actions the agents took. We refer to this data as the game's *state*.

RoboCup supports full 11-a-side games which follow the usual rules of Football, as well as several smaller sub-games that provide different challenges for agent design and control. We investigate learning by observation in the full-game scenario, but also the *keepaway* sub-game [82], described in the following section.

## 4.2 Benchmark Tasks

### 4.2.1 Keepaway

The keepaway subproblem takes place in a restricted central region of the pitch and features two competing teams: one trying to maintain control of the ball (the keepers), the other trying to gain control of the ball (the takers). Figure 4.1 shows a typical 3v2 keepaway scenario. The

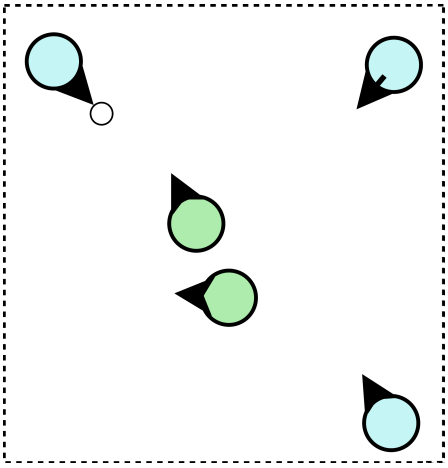

Figure 4.1: A 3vs2 Keepaway Scenario

*keeper* team is situated around the borders of the play area, with the opposing team in the centre. The game is split into temporal slices called *episodes*, where a single episode continues so long as the keeper team controls the ball. As soon as control is lost to the taker team (or the ball leaves the keepaway area), a new episode starts.

The problems associated with agent control are smaller, more restricted versions of those found in the full game. Players on the keeper team must decide when to pass the ball to a teammate, or hold on to it, while maintaining control from the taker team. The taker players must distribute themselves to intercept potential passes and tackle keeper players.

In this work we make use of existing, third-party benchmark agents that play the keepaway task[1] [82, 83]. These agents follow hard-coded behaviours, and provide a baseline level of performance, so can be considered experts in the domain. We also experiment with human controllers for these players, along with the synthetic.

### 4.2.2 Full Game

The full-game scenario in RoboCup Soccer mirrors that of traditional 11-a-side Football. Two teams compete for a fixed period of time – 6000 simulator steps — with the victor being the team that scores the most goals, as shown in Figure 4.2. While in the Keepaway subproblem,

[1] `http://github.com/tjpalmer/keepaway`

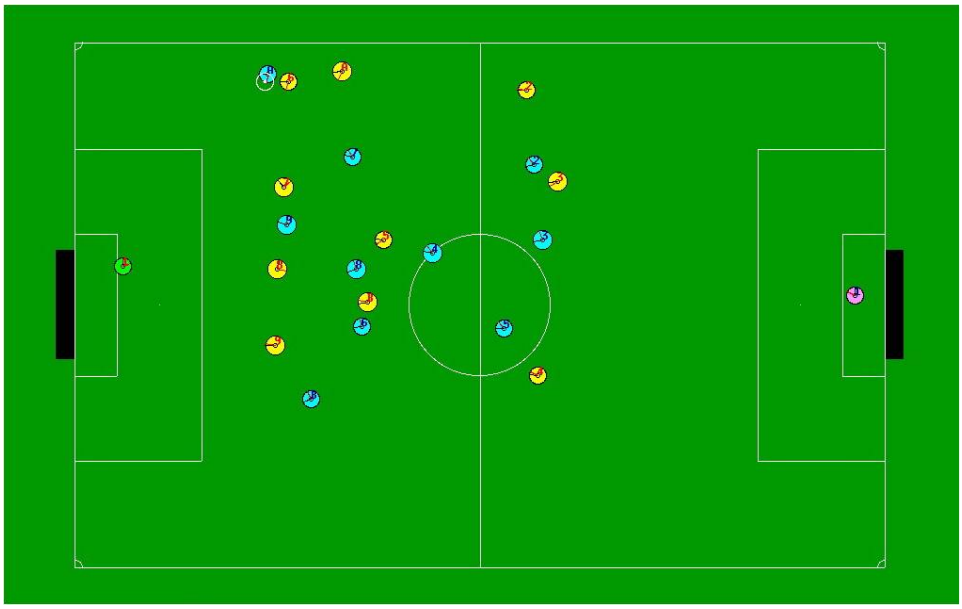

Figure 4.2: Two teams competing in the full-game scenario. Fron: Shanghai University RoboCup Project `http://robocup.shu.edu.cn/`

agents are typically homogeneous on each side, in the full-game it is more common for agents to be designed to take on various specialised team roles, such as goal keeper, striker etc.

Agents competing in this task range from those employing simple reactive controllers, to those that implement planning, or complex layered learning systems to learn on-line during games. Teams compete annually in the RoboCup 2D Soccer Simulation League, and we selected ten of the eighteen teams that competed in the 2012 league as representative of the range of state-of-the-art agents [1], including the finalists of the tournament. The following section provides a brief overview of each of the teams we selected, and the particular approaches they take, based on the Team Description Documents provided by the authors. The full list is shown in Figure 4.3.

[1] `http://wiki.robocup.org/wiki/Soccer_Simulation_League`

## 4.3 Domain Experts

### 4.3.1 Keepaway Benchmark Agents

The benchmark Keepaway agents we employ feature hard-coded behaviours for both keeper and taker teams. During play, on the taker team the two players closest to the ball will move towards it and attempt to gain control of it, with any remaining takers attempting to block the passing lanes of the keepers. For the keepers, a player will hold on to the ball so long as no takers are within a specified distance, and will pass the ball to the teammate they have the clearest shot to when takers are to close.

While these behaviours are simple, they provide the skills necessary for a player to succeed at the keepaway task whether playing as a keeper or a taker. The interactions between players employing these behaviours, and the environment itself, produces complex, emergent scenarios.

### 4.3.2 Human Controllers

In addition to working with benchmark software agents, we also gathered a dataset from human operators controlling a team of RoboCup players in real-time. Using a standard game controller, human operators were able to move players around the game environment and interact with it from the birds-eye view provided by the RoboCup Soccer Simulator visualiser, similar to that shown in Figure 4.1.

### 4.3.3 Full-Game Teams

We selected a range of state-of-the-art full-game RoboCup systems to be the subject of our experiments. While the specific approaches, architectures and algorithms employed by these systems vary widely, they all commonly attempt to solve similar problems associated with player control in the RoboCup domain. These problems range from things such as behaviour selection for selecting attacking or defending roles, marking opposing players, evaluation of the success potential of shots or passes, and lower-level skills such as dribbling or moving to evade

opponents.

This section introduces each of the full-game teams we used, and gives a brief overview of the techniques and tools used to control the team. These are taken from the technical reports written by each team to describe their approach.

| Name | Origin |
|---|---|
| AUA2D [99] | Anhui University of Architecture, China |
| AUT [51] | AmirKabir University of Technology, Iran |
| Axiom [32] | University of Science and Technology, Iran |
| Borregos [52] | Tecnolgico de Monterrey, Mexico |
| CSU Yunlu [90] | Central South University, China |
| DreamWing [98] | Anhui University, China |
| GDUT TiJi [36] | Guangdong University and Technology, China |
| Helios [3] | Fukuoka University, Japan |
| LegenDary [72] | Islamic Azad University of Lahijan, Iran |
| WrightEagle [8] | University of Science and Technology, China |

Figure 4.3: List of RoboCup 2D Full-game teams used in our experiments.

**AUA2D** Players in the AUA [99] team utilise an algorithm that determines whether they will shoot, pass or dribble the ball. The algorithm produces a measure of confidence in the range $[0, 1]$ called the *Shoot Value*, an estimate of how successful a shot at the opposing goal will be, based on various spatial features the player observes. When in posession of the ball, based on its own shoot value and an estimate of the shoot value of its observable teammates, the player determines whether to shoot at the goal itself, pass to a teammate with a higher shoot value, or dribble the ball into a more advantageous position – a point on the field where the player's shoot value would be improved. The decisions being based on thresholds on the shoot value set in advance.

For passing, the players implement a probabilistic approach that attempts to create a passing path between observable teammates, by maximising the probability of each team-mate in turn

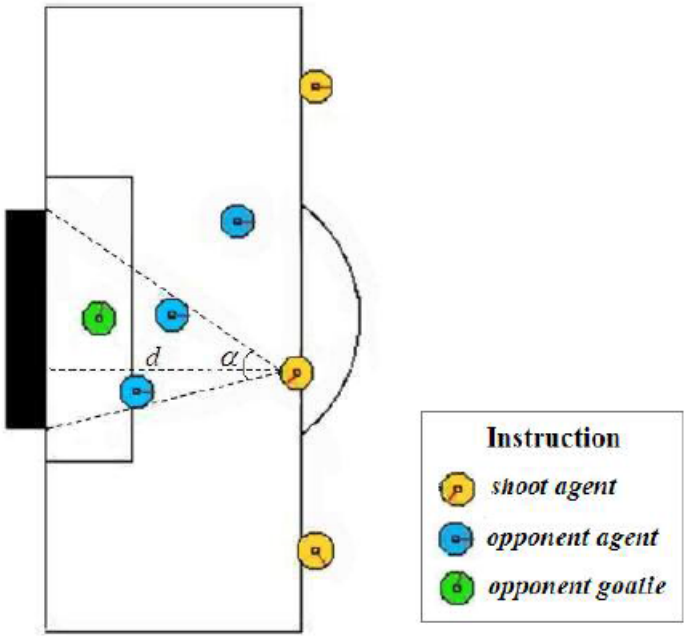


Figure 4.4: Factors considered in estimation of Shoot Value [99]

passing to the next.

**AUT** Players in the AUT [51] team implement a number of algorithms for offensive and defensive skills. AUT utilises an algorithm to rank players on the opposing team to determine which are the most important to mark, and to assign team-mates to mark specific opponents. This importance measure is based on spatial information, such as the distance between players, the strategic importance of a particular area on the pitch (for instance, it may be more important to mark opposing players closer to one's goal than further away), and the disturbance a candidate marking would make to the existing formation. This assignment is performed by the goal-keeper, which informs other players of their marks through a communication channel. The goalkeeper uses a similar algorithm for assigning defenders to block attacking, dribbling players.

Offensively, players individually utilise an algorithm to escape through the opposing defending line taking into account spatial features such as their distance from the ball, distance from the goal, distance from formation position and the opposing defenders distance. This pro-

duces a score for areas on the pitch, with the players moving to those areas that provide the highest scores.

Similarly to the AUA players, the AUT agents employ a decision-tree like mechanism to determine whether to shoot, pass or dribble the ball, based on thresholds applied to score calculations from observed spatial features.

AUA features explicit communication between players. When defending, the goal-keeper agent ranks nearby friendly players based on an estimate of their ability to stop an incoming attack, based on their positions and movement angles, and assigns each player in-turn to attempt to block an incoming attack.

**Axiom** The Axiom [32] players employ a Genetic Algorithm which treats the search for a particular position on the pitch with a high shoot value as an optimisation problem. It considers angles created by goalpost lines, the distance that the ball would traverse to the goal, the distance between the shooter and the nearest defender, and the angular distance between the shooter and the nearest defender. The GA searches for positions that provide a high probability (based on a pre-selected threshold) of success when shooting at the goal from a particular position.

In addition, the system employs a Neural Network based approach to pass selection, to predict the result of a pass to a particular team-mate. The system is trained on a set of example scenarios devised by the authors, to maximise the probability of success in particular scenarios. The selection between different strategies – to shoot, pass or dribble – is controlled by a Markov Decision Process. Based on spatial information, such as self position, opposing/friendly players position and velocities, the system selects the optimal strategy (based on a set of training examples provided by the authors) in each state.

**Borregos** The Borregos [52] players employ a mix of reactive controllers, based on engineered heuristics, and multi-agent programming. Borregos appears very much to be an example of a team benefiting from much knowledge engineering. Players are guided by heuristics that guide high-level action selection to help gain and maintain control of the ball, these heuristics are not made available by the developers, and so we are unable to discuss them in any great

detail. The performance of the Borregos team is typically average, being found in the middle to lower rankings in tournaments.

**CSU Yunlu** The CSU Yunlu [90] players employ a set of predefined reactive plans which are executed when particular game-state situations are encountered. The execution of these plans is predicated on observed spatial information, such as the position and angle of a goalkeeper in relation to the goal when considering a strike, or the speed and velocity of an opposing player when attempting to intercept them.

The system uses a formation management algorithm when players have ball possession, such that players will move in the direction pointing towards the opposing goal that has the least number of opposing players around it.

**DreamWing** Players in the Dreamwing [98] team make use of Q-Learning in order to maximise a reward function during gameplay. This reward function is calculated on-the-fly based on several hand-coded heuristics and weightings on state information, such as ball position and motion, as well as opposing and friendly player positions. During play, this is used to execute what is determined to be the most optimal behaviour – from shooting at the goal, passing the ball, dribbling etc.

**GDUT TiJi** GDTUT Tiji players [36] utilise reactive controllers engineered by hand to produce specific behaviours in certain game states. These game states are based on the observed positions of the ball, opposing players and friendly players. For instance, players attempt to avoid being marked by opponents by moving in such a way that ensures no opposing players are located within a pre-set distance threshold.

**Helios** The Helios system [3] is a champion RoboCup system (the winner of the 2012 competition) which utilises complex, layered on-line planning to search the continuous state-space of the game for the optimal policy to pursue during gameplay. To do this, it makes use of an extensive number of hand-coded and finely tuned heuristics.

**LegenDary** LegenDary players [72] utilise a scoring mechanism for ranking players to pass the ball to, similar to the shoot value used by AUA [99], and is also based on the spatial positioning and motion of the ball and nearby players. When dribbling, the system also relies on several pre-defined paths across the pitch, that are evaluated and chosen based on the proximity of opposing players to those paths.

**WrightEagle** The WrightEagle system [8] is another champion RoboCup system, and the runner-up in the 2012 competition. The system employs on-line formation detection, which classifies opponent formations and allows the system to formulate counter-strategies. In addition, WrightEagle agents make use of Monte-Carlo Localisation, which allows them to represent and reason about global positional information, rather than local, relative information, as is most common for RoboCup players. Further, a Monte-Carlo algorithm is used to sample the space of potential sequences of movement and passing actions over a finite time horizon.

# Starcraft

## 5.1 Introduction

Starcraft is a Real-Time Strategy game released by Blizzard Entertainment in 1998[2]. RTS games typically involve two or more players initially situated in different locations on a two-dimensional landscape, which may have a variety of geographical features such as hills, ramps, valleys, as well as navigable and unnavigable areas. Players start with a selection of rudimentary structures and units, which must be used to gather nearby resources, which are finite. These resources are used to produce new units, building and upgrades, with the aim of producing military units to destroy the opponent's base and win the game. This is accomplished by selecting build options from various buttons and menus. Military units come in a wide variety of types, differing in speed, armor, movement type (land, or air), thus bearing their own strengths and weaknesses, and are typically segregated according to a player's faction (or race), resulting in an asymmetric game if each player picks a different faction. RTS games are also played under the principle of *partial observability* as each player can only observe the area of the environment directly surrounding their own units and buildings, with the unobservable area being covered by what is called the *fog of war*. Starcraft is by no means the only RTS, some others include Total Annihilation, Age of Empires, Dune, Warcraft and the Command & Conquer series, among others. Starcraft is however the most popular RTS in the world, both in the professional gaming community as well the AI community. In the gaming community, millions of dollars a year are won in Starcraft tournaments around the world, with many big technology companies such

[2]http://us.blizzard.com/en-us/games/sc/

as Fujitsu and Samsung sponsoring their own professional teams. Starcraft is recognised as a sport in South Korea, and taken very seriously, with several individuals being sentenced to jail time for match-rigging in 2010 [61]. In the AI community, each year the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment hosts the Starcraft AI Tournament, which pits Starcraft AI systems designed by teams of AI researchers, or individuals, against each other in a tournament very similar to the annual RoboCup tournament [15]. There are now several similar tournaments occurring worldwide each year, with an ever-growing roster of entrants. It is Starcraft's position as the *gold standard* of RTS games played at the highest level of human competition that makes it an attractive and interesting platform for AI research. Due to its age (17 years at time of writing) many instances run on any modern computer at the speeds needed for batch processing. . The game is considered to be supremely well-balanced and robust [16], with exploits and bugs being all but eradicated over the last 17 years of patches from the developer, a level of maturity that competing platforms cannot offer. This confidence in the platform is required to maintain the huge financial investment that the game's tournaments see each year.

RTS games like Starcraft require a range of concurrent skills in order to be successful, such as goal-directed planning, reasoning under uncertainty and creative adaptation, often requiring continuous input and, for human players, management of limited attentional resources. This must be accomplished in real-time, which exasperates many of the already difficult AI challenges present. In recent years the game has become a focus of interest from the AI community due to the advent of the Brood-War API (BWAPI[1]), a software API which allows for the injection of code into the Starcraft game engine, and facilitates the production of third-party AI players that allows an engineer to design a system to interact with the game in the same way a human player would. In principle, AI players have significant advantages over human players through being able to precisely and concurrently control a far greater number of units, and to control an unlimited number of forces that may be spread out and separated from one-another, a task where humans are limited by their own attentional focus. Simultaneously, an AI system

[1] http://code.google.com/p/bwapi/

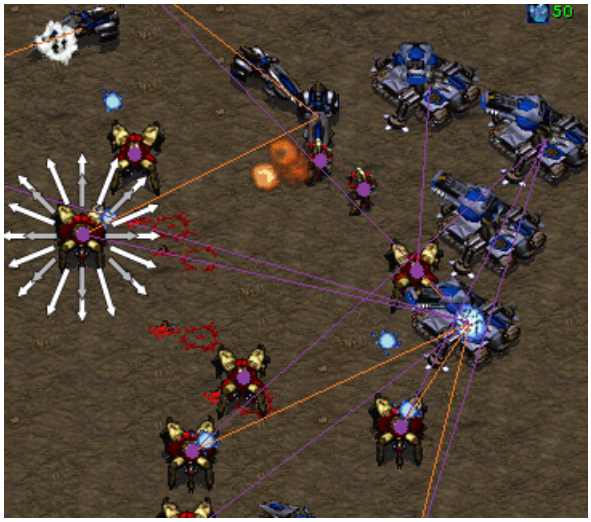

Figure 5.1: A combat scenario in Starcraft, taken from Synnave and Besserie [84]

should be able to control units individually with a far greater degree of accuracy and precision than a human player, and exploit their strengths accordingly. The question of how to practically accomplish this theoretical level of performance has sparked a great deal of research interest in the topic.

In Real-Time Strategy games, the overall skill of a player is typically thought of as being based on their competency in two broad areas: micro-management and macro-management. Macro-management encompasses high-level tasks, such as training units, building structures, selecting upgrades, and scouting relating to a strategy [16]. Micro-management on the other hand concerns the task of managing single units, or groups of units, and issuing tactical commands during combat, such as movement and attack commands. Good micro-management skills are those which best exploit the characteristics of the units available to the player, the environment, and potential weaknesses of the opposing units – for instance, a faster unit leading a slower opposing unit. Starcraft features asymmetry inbetween the factions themselves and the units available to each faction. This asymmetry is considered to be extremely well-balanced, as mentioned previously, with no single global, dominating strategy. A combat situation involving units of various types is shown in Figure 5.1

Units are controlled in Starcraft by selecting either a single unit or a group, and clicking a point in space to move to, or an enemy unit for them to attack. Units may also have special

abilities that are deployed in a similar way after being selected. Starcraft features three distinct playable races – the Terrans, the Zerg and the Protoss – each having a dozen or so different units available to them ranging from land units, air units and support vehicles.

. That is, at runtime a system would select the most appropriate micro-management behaviour from a library of pre-defined behaviours and load these on to its units, with these behaviours being either learned or pre-defined by an engineer. This prior work motivated us to apply the results of our work with RoboCup Soccer to the Starcraft domain, to determine if micro-management behaviours could be learned as a way to populate this library of micro-management behaviours.

## 5.2 Benchmark Agents

While in the RoboCup domain we were able to make use of existing third-party benchmark agents, for the Starcraft domain no such agents existed for the tasks we selected. As such, we first selected a set of standard, pre-existing sub-tasks in the Starcraft domain – much like the RoboCup Keepaway task – and developed our own benchmark agents, publishing our work and making it available for others to build on [96]. We built the benchmark agents using the same rule-based and sampling-based approaches we developed in a side-project [97], which had previously competed in international Starcraft AI competitions. No modifications were made to the underlying agent controllers, though a pathfinding algorithm was added to allow the group of units to navigate through the corridors of the third task.

The benchmark system largely relies on an implementation of a hit-and-run tactic. On each frame, the agent samples the environment using a point cloud, and heuristically evaluates each point for a relative safety value based on the proximity of friendly and enemy units to that point. If the agent is near a point it considers safe, it will attack any nearby enemy, if not, it will move away towards a safe space. Variations of this tactic are included for different unit types in order to take advantage of their specific characteristics. This produces complex, emergent behaviour in groups of agents at a low computational and engineering time cost. A video of the system

are available online [1]. In the video, the Purple, Protoss player is controlled by our benchmark system, and fends off an early-game attack from the Blue player's melee units, with much co-operation between units, though none was pre-programmed. While we make no claims that these benchmark agents behave in any *optimal* way with regard to the benchmark tasks we select, they do however fit our definition of what constitutes an expert, and achieve a baseline level of performance.

### 5.2.1 Human Controllers

To study our ability to reproduce observed human behaviour, we gathered Human subjects who possessed minor prior familiarity with the Starcraft domain. No information on the tasks was given beyond the instructions included with the training scenarios, and Human controllers remained unprompted during the course of each experiment. For each of the five test subjects we gathered 30 episodes of data per task, for a total of 150 episodes of data per task.

## 5.3 Benchmark Tasks

In our experimentation with Starcraft we selected three benchmark scenarios from a set of training maps developed to allow players to practice particular micro-management skills[2]. These include a kiting task, a group micro-management task, and a final, more difficult task involving a sequence of group combat scenarios of increasing difficulty and opponent make-up.

#### Task One – Kiting Task

In real-time strategy games, kiting is a tactic whereby a unit will quickly attack a pursuing target before immediately retreating, taking advantage of differences in movement speed or attack range to stay out of danger.

[1] https://vimeo.com/45580695
[2] http://wiki.teamliquid.net/starcraft/Micro_Training_Maps


Figure 5.2: A combat scenario in Starcraft

Figure 5.3: An example of a Kiting scenario

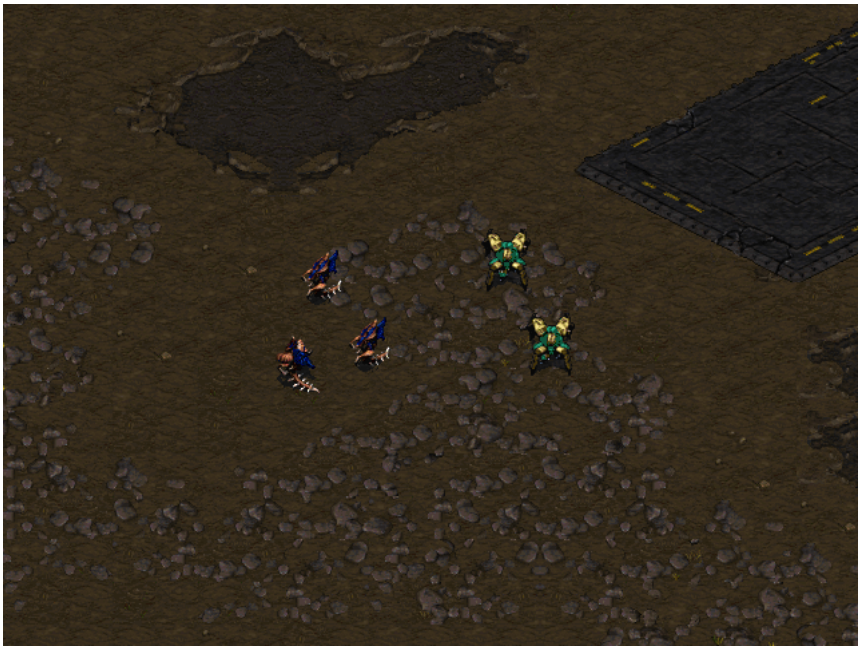

Figure 5.4: The 3vs2 scenario which is the focus of the experiment of Parra[60]. The Dragoons (in teal) move faster than the Hydralisks (in blue), but the Hydralisks deal far more damage to the Dragoons than vice-versa, and also outnumber them, meaning that a focused attack from all three would destroy a Dragoon almost instantly.

#### Task Two – 3vs2 Combat

In our second task, we repeat the experiment of [60]. In this task, the player is given control of two Dragoons (large, ranged units) and is tasked with defeating a squad of three Hydralisks from the Zerg race (medium-sized ranged units). The player is not only at a disadvantage in terms of force sizes, but also underlying game mechanics. In Starcraft, units deal less damage to targets that are smaller than them. The Hydralisks deal 100% damage to the Dragoons, whereas the Dragoons only deal 75% damage to the Hydralisks. While the Dragoons do more base damage per hit than the Hydralisks (20 vs. 10) the Hydralisks attack twice as fast as the Dragoons. The player must make aggressive use of the advantages of their own units by splitting their force up, and ensuring that the enemy units cannot focus fire on a single unit. In addition, kiting tactics could be used, but this may be difficult for a human player since it would require controlling multiple units moving in different directions simultaneously.

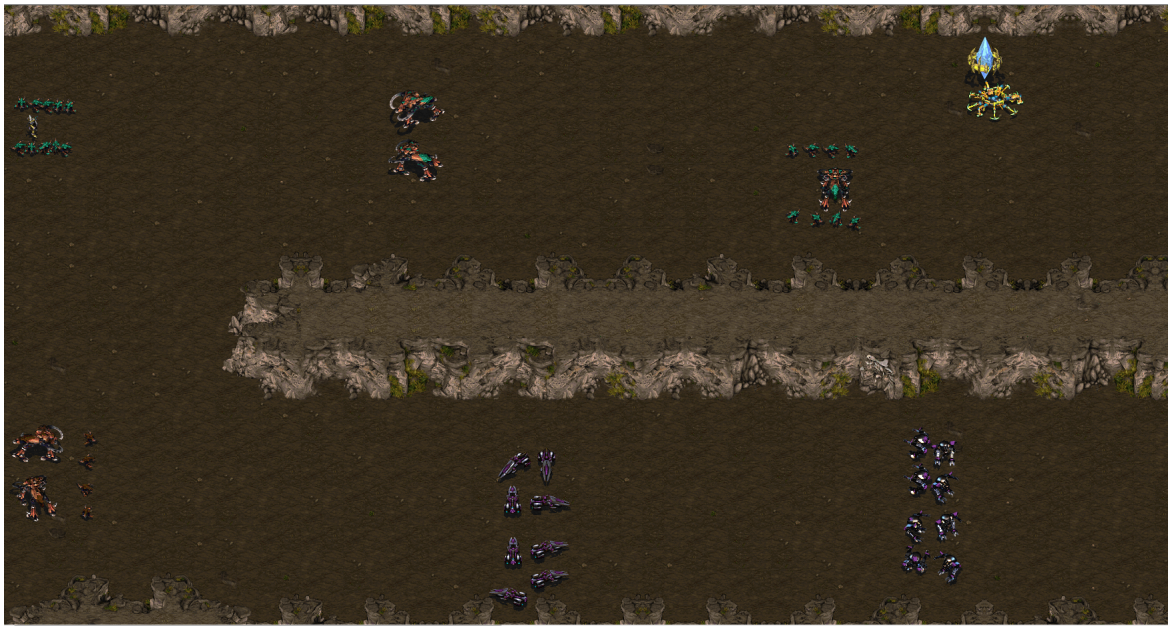

Figure 5.5: Two corridoors along which the agent's army must move in the *The Fall of the Empire* scenario

### Task Three – Group Combat Sequence

In the final scenario, we consider a more difficult task – a scenario named *The Fall of the Empire*. Initially a player is given access to four Dragoon units, and must progress around a series of corridors, defeating groups of enemies from each of the races in Starcraft along the way. The player is occasionally afforded reinforcements at the end of each corridor. The map is split into four corridor, each featuring combat scenarios of increasing difficulty, with the ultimate goal being to reach the end of the map. This scenario tests much larger-scale combat than the previous, as well as combat with a more varied array of units.

# Experiments

This chapter is the first of our two experiments chapters, and presents our work in the Starcraft and RoboCup (Keepaway and Full Game) domains focusing on *imitation*. Each sub-section follows the same structure for experiments, with our evaluation of a particular domain having two distinct parts: *learning* and *deployment*. We use the best performing learning approaches (according to standard metrics) for our deployment experiments, which are in turn evaluated on task-specific metrics. We briefly discuss the results in each section, but leave deeper analysis and comparison between domains for the next chapters.

## 6.1 Experimental setup

## 6.2 A note on deployment experiments

While the approach to representation and learning used in our work is typically unchanged between domains, the result of a learning algorithm must ultimately be integrated into a system capable of controlling an agent in a target domain. This inevitably requires domain-specific machinery, must of which we discussed in the previous chapter, but we will also discuss others in this chapter as it becomes necessary.

## 6.3 Experiments with RoboCup Keepaway

### 6.3.1 Learning Predictive Models

Using the benchmark agents of [82] we generated datasets for three common keepaway configurations – 3vs2, 4vs3 and 5vs4 – producing 1000 episodes per configuration. In the raw data extracted from the RoboCup simulator for this task, the behaviour traces contain the action chosen by each agent on each time step along with their continuous parameters (e.g. $kick(64)$ or $dash(99)$), as well as the metric state of the world. We discard the continuous parameters from the actions, leaving only the action class, similar to the approach used by [27]. This makes the classification/prediction problem easier, allowing us to reconstruct action parameters as described in the previous section. The metric observations are transformed into a set of QSR features using the calculi described previously, along with our representation selection algorithm. We use this to produce a model which predicts the action an agent will take given the state of the world it can observe, treating the learning problem as one of classification.

Comparing the performance of classifiers is not as straightforward as just looking at their prediction accuracy. The data features significant class imbalance – the balance of kick, dash and turn actions being 11%, 22% and 66% respectively. Figures 6.1, 6.2 and 6.3 show a comparison between three different learning algorithms – a Bayesian Network, the C4.5 decision-tree classifier and a Support-Vector Machine – using both the underlying metric and automatically-configured QSR representations, evaluated on ten-fold cross-validation in each of the three scenario configurations. The implementations of the learning mechanisms were taken from the Weka toolkit [37], and hyper-parameter optimisation was performed as described in the previous section in 1. The inner loop of this algorithm executes the hyper-parameter optimisation step, with the tools being supplied by the Weka toolkit. For the SVM this resulted in the use of a radial basis function and $C$ of 1.0. For the C4.5 classifier a confidence factor of 0.27 was used.

The results show that we are able to predict these discrete actions with a high degree of

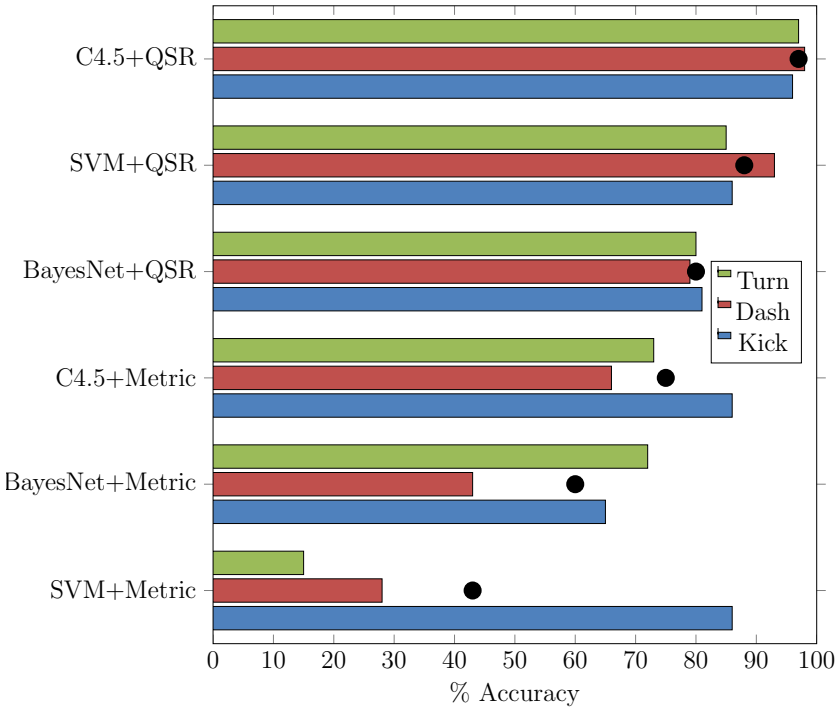


Figure 6.1: Per-class performance of classifiers in the 3vs2 scenario with benchmark agents. Black dots indicate the average performance of a given classifier.

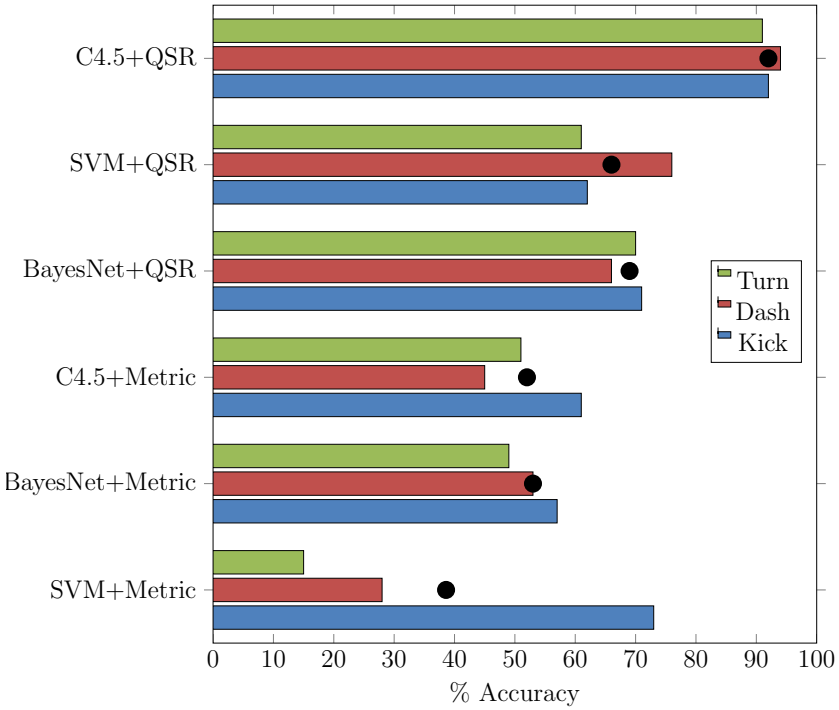


Figure 6.2: Per-class performance of classifiers in the 5vs4 scenario with benchmark agents. Black dots indicate the average performance of a given classifier.

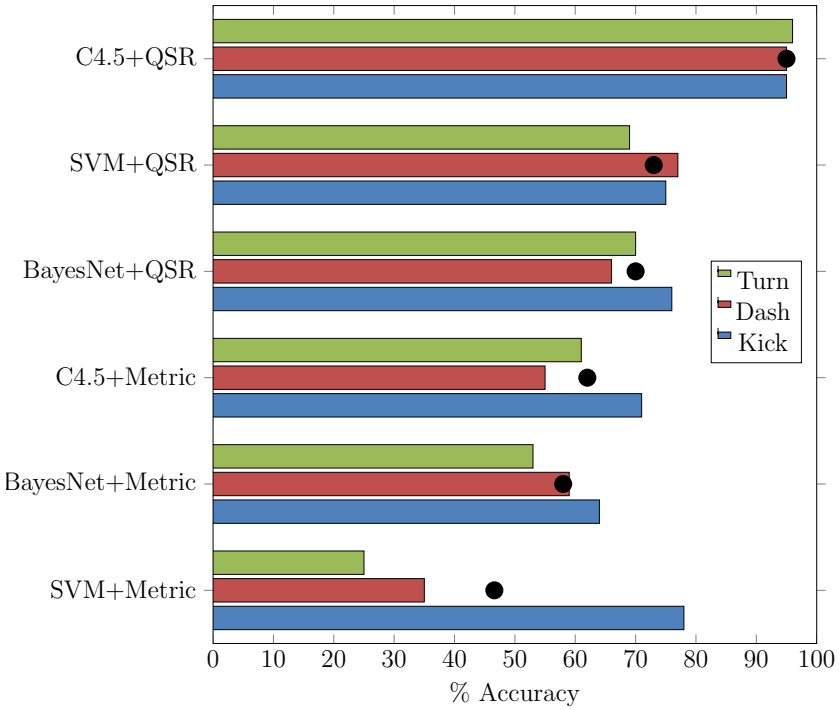


Figure 6.3: Per-class performance of classifiers in the 4vs3 scenario with benchmark agents. Black dots indicate the average performance of a given classifier.

accuracy (97% for the C4.5 classifier) when utilising the QSR-based representation scheme and self-configuration. Utilising the raw metric data provides generally worse results for all learning algorithms. Some task structure is able to be detected, even by the algorithms that perform generally poorly – for instance, the $kick$ action is typically the most easily predicted action, as is it is one that has a very discriminative set of states – the ball must be within a set range of the player before this action can be executed. We will discuss this further later on.

### 6.3.2 Learning Predictive Models From Humans

In addition to working with benchmark software agents, we also gathered a dataset from Human operators controlling a team of RoboCup players in realtime. Using a standard Xbox 360 game controller, Human operators were able to move players around the game environment and interact with it from the birds-eye view provided by the RoboCup Soccer Simulator visualiser. This is in contrast to the noisy and limited field of view provided to synthetic agents. The Human experiments also differed in another key way, in that Human controllers were only able to select

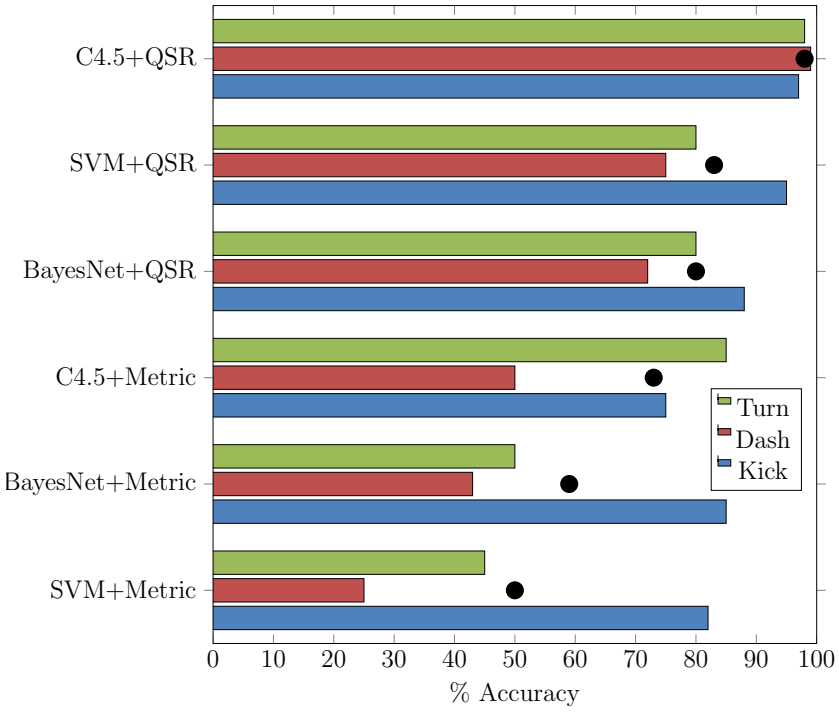


Figure 6.4: Per-class performance of classifiers in the 3vs2 scenario with Human controllers.

from a set of actions with pre-defined parameters. While software agents performed actions with real-valued parameters, the Human operators could only choose from a discretised action set with fixed parameters, e.g. a weak, medium or strong kick, all taking their direction from the way the player currently faced. No prompting was provided to the participants during the study.

Our data set for this experiment is smaller and focuses on a single task, due to the time expense of collecting Human data. We collected 200 episodes of the 3vs2 game configuration, wherein three Human players simultaneously controlled members of the keeper team. The taker team was controlled by the benchmark agents used in the previous evaluation. We then built predictive models in same way as before. The predictive results are shown in Figure 6.4, evaluated again using ten-fold cross-validation. Our QSR approach again outperforms the metric representation in most cases, and demonstrates that QSRs can encode Human behaviour as well as that of artificial agents, and that a decision tree can provide a strong predictive model for that behaviour. Again we see very similar structure between algorithms in terms of the relative success at which they predict each action, which we find in all of our results in RoboCup Soccer.

| System | Rep. | Avg. Episode Length |
|---|---|---|
| Human Controller (Single Best) | - | $186 \pm 8$ |
| LbO (Human, Single Best) | QSR | $164 \pm 10$ |
| Human Controller (All) | - | $156 \pm 14$ |
| LbO (Human, All) | QSR | $134 \pm 17$ |
| Benchmark Agents | - | $131 \pm 18$ |
| LbO (Benchmark) | QSR | $121 \pm 6$ |
| LbO (Human, Single Best) | Metric | $82 \pm 25$ |
| LbO (Human, All) | Metric | $76 \pm 32$ |
| LbO (Benchmark) | Metric | $65 \pm 37$ |
| Random | - | $60 \pm 20$ |

Figure 6.5: Average episode duration in simulator steps showing variation in performance after learning from both Human and benchmark agents. C4.5 is used for both metric and QSR-based approaches.

This suggests to us that there are definite spatial features that provide information about specific actions in given states, and our QSR system is able to determine these features. One example that shows this is that, if we measure the average distance between a player and the ball when the player executes a $kick$ action, the value is $1.03m$ – that is, Human players typically wait for the ball to be within $1.03m$ before kicking it. Our learning agent discovers this, and divides its representation of qualitative distance into equally sized rings of around $1m$. It additionaly divides its representation of qualitative direction into increments of $13$ degrees, as this allows it to focus its front on the ball accurately as it, similar to the way Humans do, as it approaches.

### 6.3.3 Deploying Models of Keepaway Agents

To test deployment of models, we ran experiments systematically replacing synthetic team players with players controlled by our system. This allowed us to evaluate the performance of the LbO agents, not only when teamed with other LbO agents, but also when teamed with the hand-coded agents in mixed teams. As the goal of the keeper team is to maximise episode duration (by keeping the ball), we evaluate our system using the metric of average episode duration over 1000 episodes.

Figure 6.6 presents the results of the systematic replacement of benchmark agents by agents deploying LbO behaviour models. Average episode length decreases consistently as LbO agents

| LbO Agents | Benchmark Agents | Avg. Episode Length |
|---|---|---|
| 0 | 3 | $131 \pm 18$ |
| 1 | 2 | $127 \pm 16$ |
| 2 | 1 | $124 \pm 14$ |
| 3 | 0 | $121 \pm 11$ |

Figure 6.6: Effects of replacement of pre-programmed benchmark agents in the keeper team with agents using behaviour models generated by observing those benchmark agents (C4.5+QSR).

are swapped in, showing that these agents cannot completely recreate the precise behaviour observed. This is largely due to misclassifications, and errors in their ability to select parameters of actions. Some misclassifications are more severe than others – missing a common action, like a turn or dash on one frame may not have a major impact, however missing a rare, important action such as a kick may. The same issue arises when selecting parameters for actions. We find that these kinds of errors have the side-effect of landing agents in states that they had not previously observed. This leads to a feedback loop: the agent will make a mistake, enter into states it is not familiar with, perform poorly, and performance will degrade. This highlights a limitation of our approach so far, and indicates a point where reinforcement learning approaches could be integrated into the process. In the next chapter of experiments we employ the SARSA [44] algorithm to explore one possible way our LbO system can be used in conjunction with RL approaches.

With our Human experiments we compiled all of the data from each participant into a single data set and used this to train our agents. An alternative approach would be to train a model on a single Human, in Figure 6.5 we show the results of an agent trained on the single best Human, which outperforms all of the other LbO agents. We selected the best Human based on the player that lost the ball the least number of times, and whose passes to their team-mates were the most commonly successful. The Human data exhibits a great degree of variance, for instance when looking at the whole dataset we see a clear learning curve in terms of performance over time as the Human subjects learned to understand and master the task, as in Figure 6.7. This is in stark difference to the synthetic agents, who maintain a level of performance roughly within the same bounds.

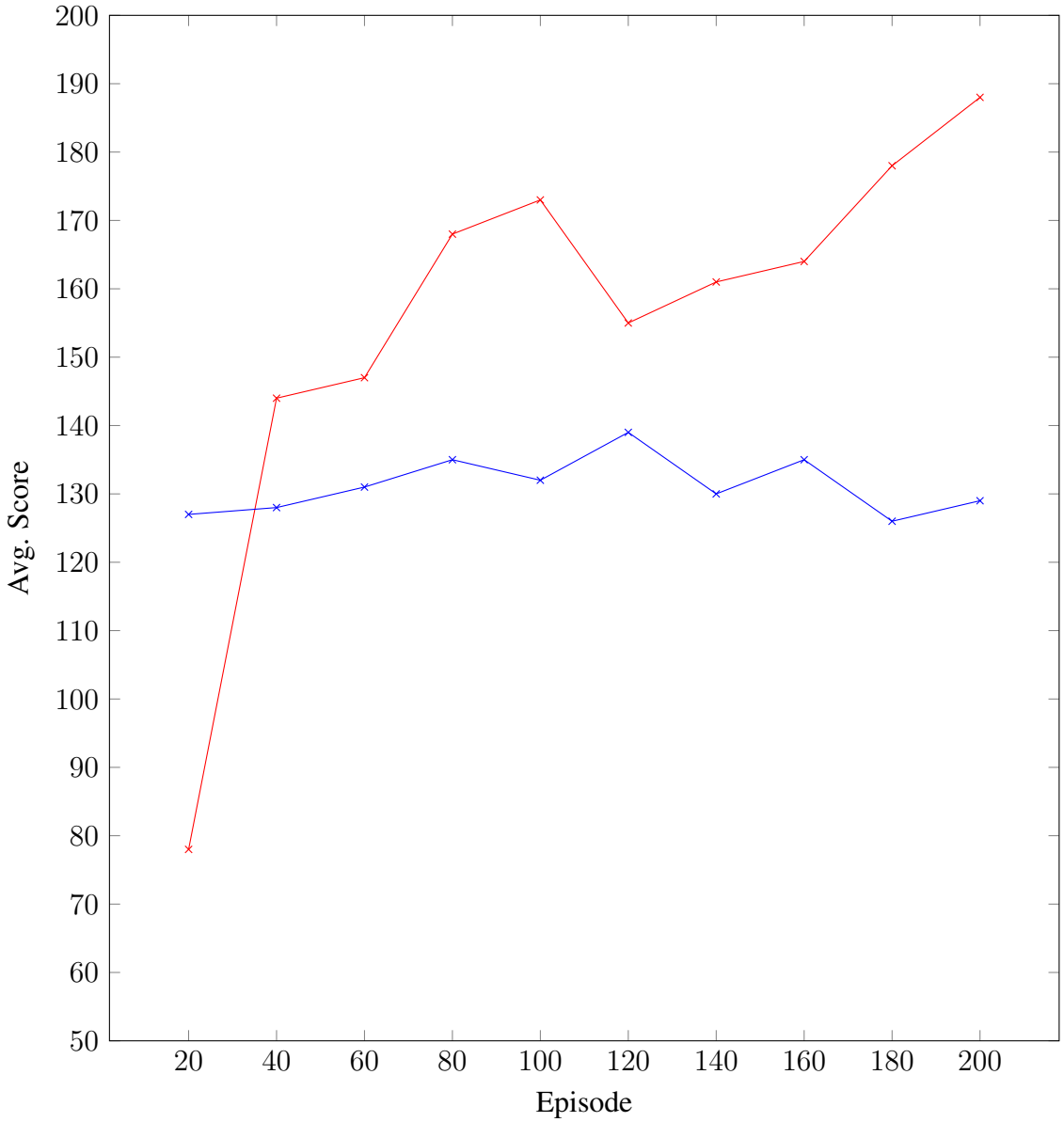


Figure 6.7: Average scores in the keepaway task for all Human test subjects (red) and benchmark agents (blue), showing that Human controllers exhibit a learning curve, whereas performance of synthetic agents falls within a static range.

| LbO Agents | Benchmark Agents | Avg. Episode Length |
|---|---|---|
| 0 | 3 | 131 |
| 1 | 2 | 121 |
| 2 | 1 | 134 |
| 3 | 0 | 155 |

Figure 6.8: Effects of replacement of pre-programmed benchmark agents in the keeper team with agents using behaviour models generated from the best Human player.

In Figure 6.8 we performed the same replacement experiment as before, but with an agent model trained on the best Human player. In Figure 6.6 we replaced the synthetic team with our learning agents, resulting in an overall decline in team performance. The overall team performance initially declines before increasing as the LbO agents are introduced – this is because the controllers behave in different ways that are not complementary, and the synthetic agents are perhaps looking for certain visual cues from their teammate that are never observed. We note that the benchmark agents always attempt to face towards the ball, however the Human controller that we trained the model on typically faced towards its *teammates*, except for when the ball is within a certain distance. Over tiem, the LbO agents begin to outnumber the original team, and the team performance picks up.

Figure 6.8 presents a comparison between the QSR and metric representations when all 3 keeper agents are replaced, and also the difference in performance of models learnt by observing synthetic and Human experts. Again the QSR representation outperforms the metric, and there is a larger drop in performance on the Human data than synthetic in terms of average episode length in simulator steps. The larger deficit on Human data can be explained by the fact that the Human experts were actually naïve users who had not played RoboCup previously. They therefore had to learn whilst playing and being observed. This is visible in the data, where Human episode duration starts low but increases after approximately 20 episodes. The models learnt from Humans include these initial episodes, and thus may include errors introduced by the expert – for instance, if the Humans pressed a button in error. Human controllers required a handful of trials before they were comfortable and competent with the simulator and control mechanism. This is not the case for the benchmark agents which perform consistently throughout. Discarding the initial 20 episodes from the Human data produces average episode lengths

of 129 and 96 for QSR and metric representations respectively (an improvement of 5 and 10). This is an interesting result, but discarding this data may not be the best option – we suspect it could be harnessed to provide an agent with a more thorough understanding of the task domain, by observing directly the process of an expert attempting to accomplish some goal.

## 6.4 Experiments with Full-Game RoboCup Soccer

### 6.4.1 Learning Predictive Models

The setup of our system for the full-game scenario does not change from the Keepaway scenario. However, the full-game scenario does include two additional possible player actions not used in Keepaway, namely *tackle* and *catch*, the latter of which typically being used by goalkeepers. The learning and deployment components require no adaptation or foreknowledge of these actions.

The training set for each of the full-game teams consists of learning traces taken from that team playing 10 full games against each of the other teams in our test set, detailed in Chapter 4, including itself (known as the Mirror Match) for a total of 100 games. A full game lasts 3000 time steps, and so the training set for each team consists of 300,000 frames of data. Later on we look at how varying this amount of training data affects end performance.

For each team we again compare different classifiers, an example of which we show in Figure 6.9 for CSU Yunlu [90], a team whose behaviour our system is able to predict very well. In general, average classifier performance is lower in this task than in the Keepaway task as the domain is more complex, both in terms of there being more players and activity on the field, but also because player architectures and behaviour are more complex themselves, as described in the Robocup chapter.

The average predictive accuracy for all full-game teams is shown in Figure 6.13, we also include the plots in Figure 6.10, Figure 6.11 and Figure 6.12 showing the performance of our system on a variety of systems. These are just a sub-set of those shown in Figure 6.13, and the full set of plots for all teams are in Appendix 1.

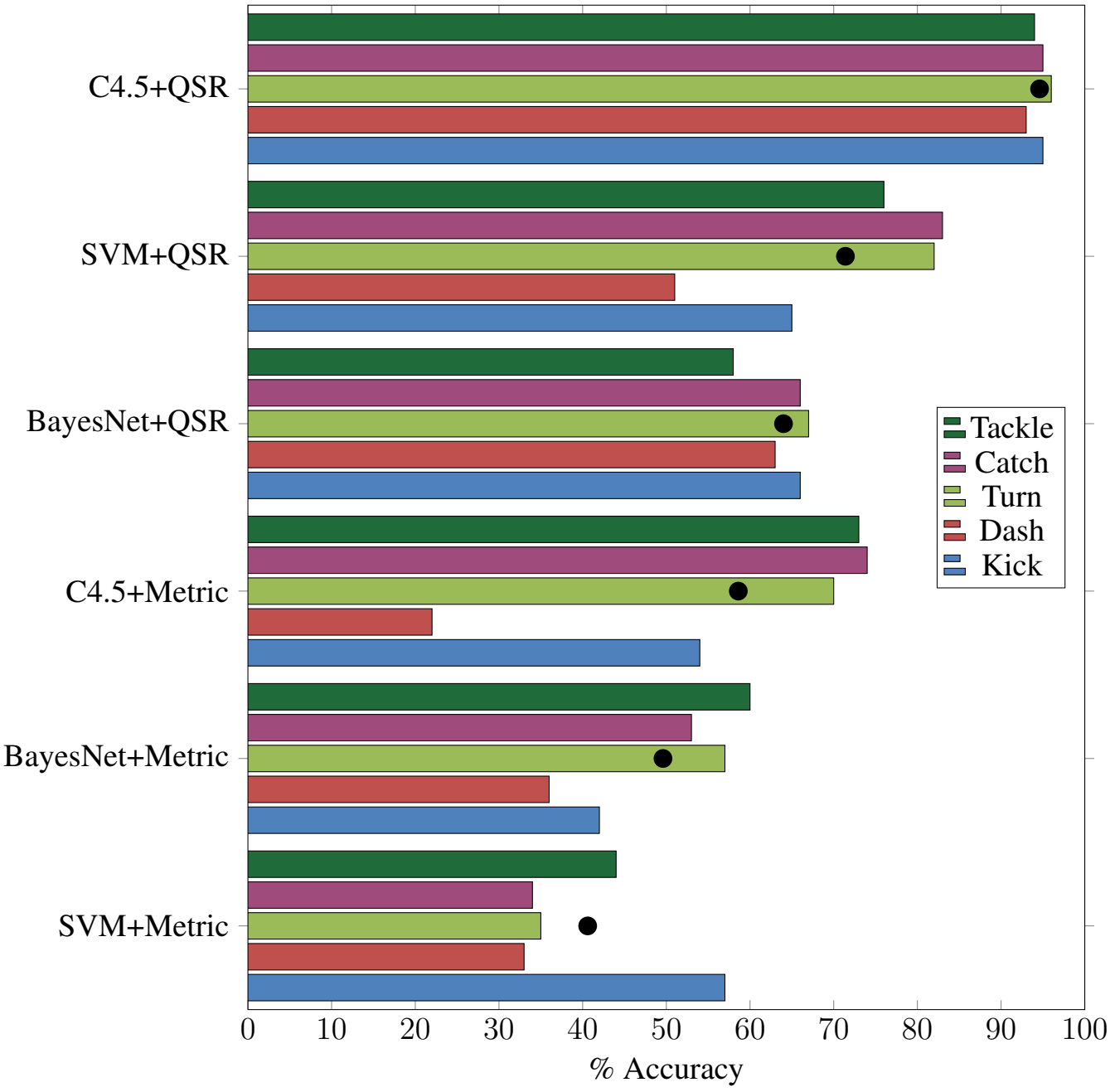
C4.5+QSR
SVM+QSR
BayesNet+QSR
C4.5+Metric
BayesNet+Metric
SVM+Metric
Tackle
Catch
Turn
Dash
Kick
0
10
20
30
40
50
60
70
80
90
100
% Accuracy

Figure 6.9:

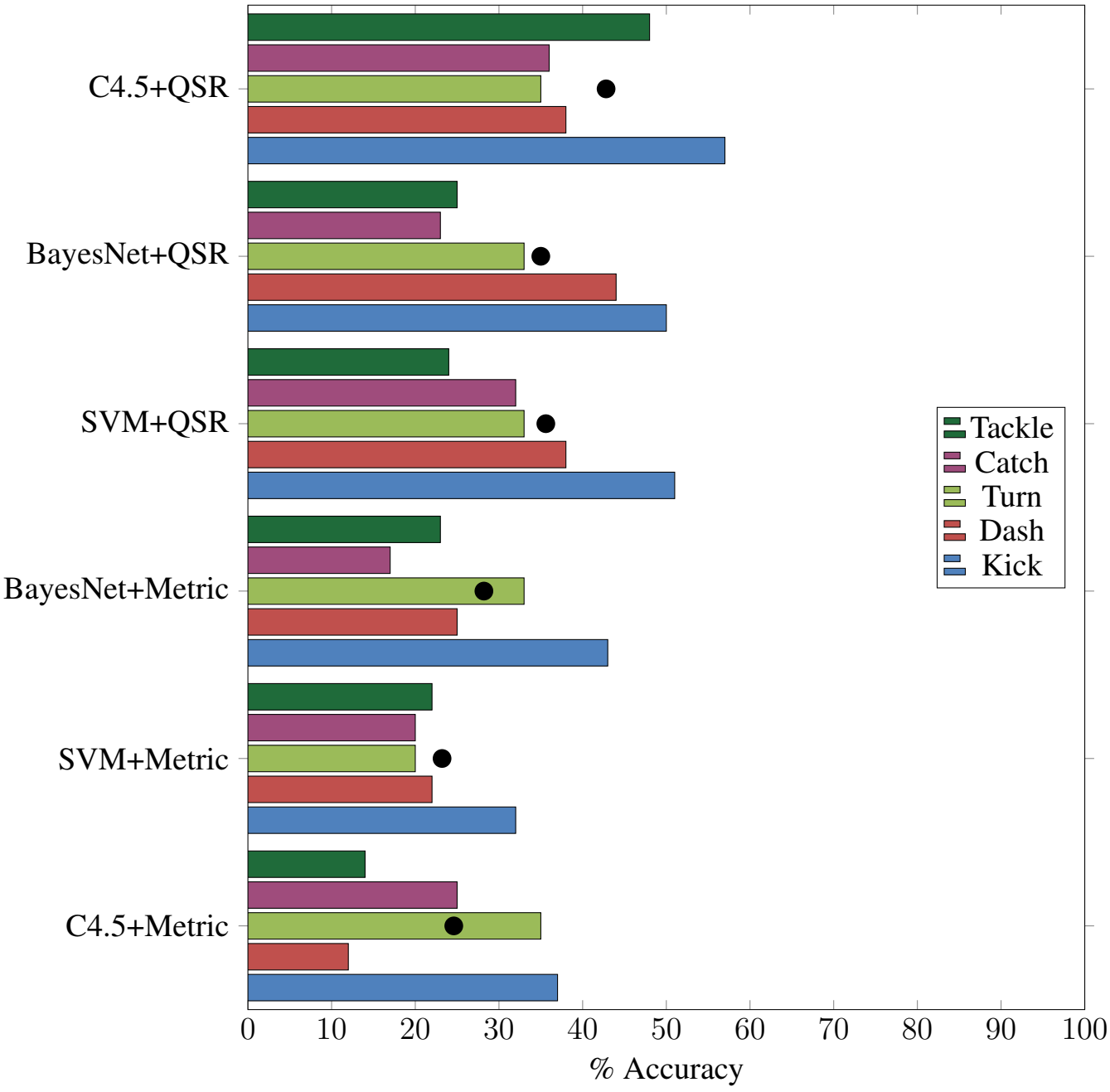
C4.5+QSR
BayesNet+QSR
SVM+QSR
BayesNet+Metric
SVM+Metric
C4.5+Metric
Tackle
Catch
Turn
Dash
Kick
0
10
20
30
40
50
60
70
80
90
100
% Accuracy

Figure 6.10:

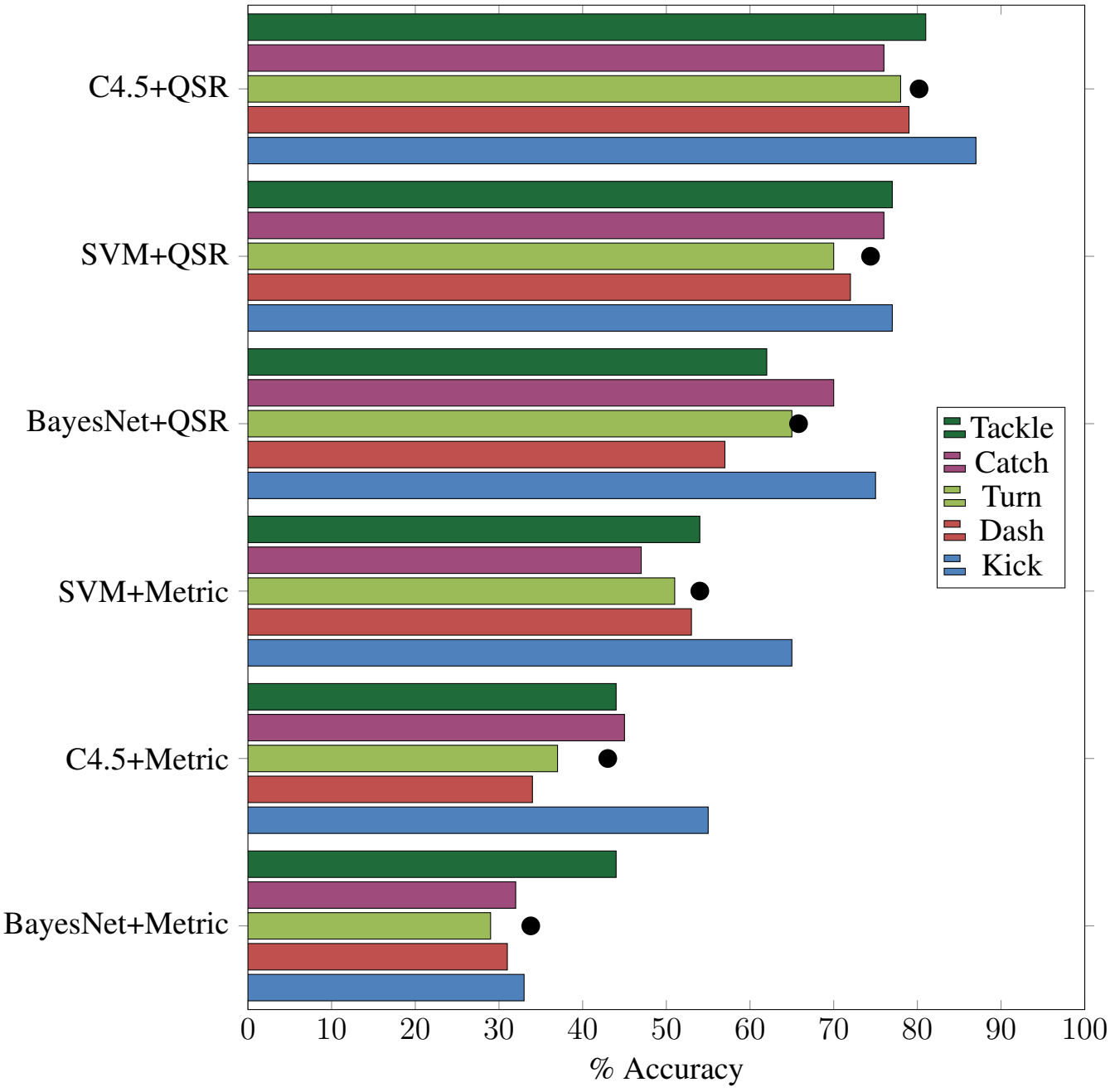
C4.5+QSR
SVM+QSR
BayesNet+QSR
SVM+Metric
C4.5+Metric
BayesNet+Metric
Tackle
Catch
Turn
Dash
Kick
0
10
20
30
40
50
60
70
80
90
100
% Accuracy

Figure 6.11:

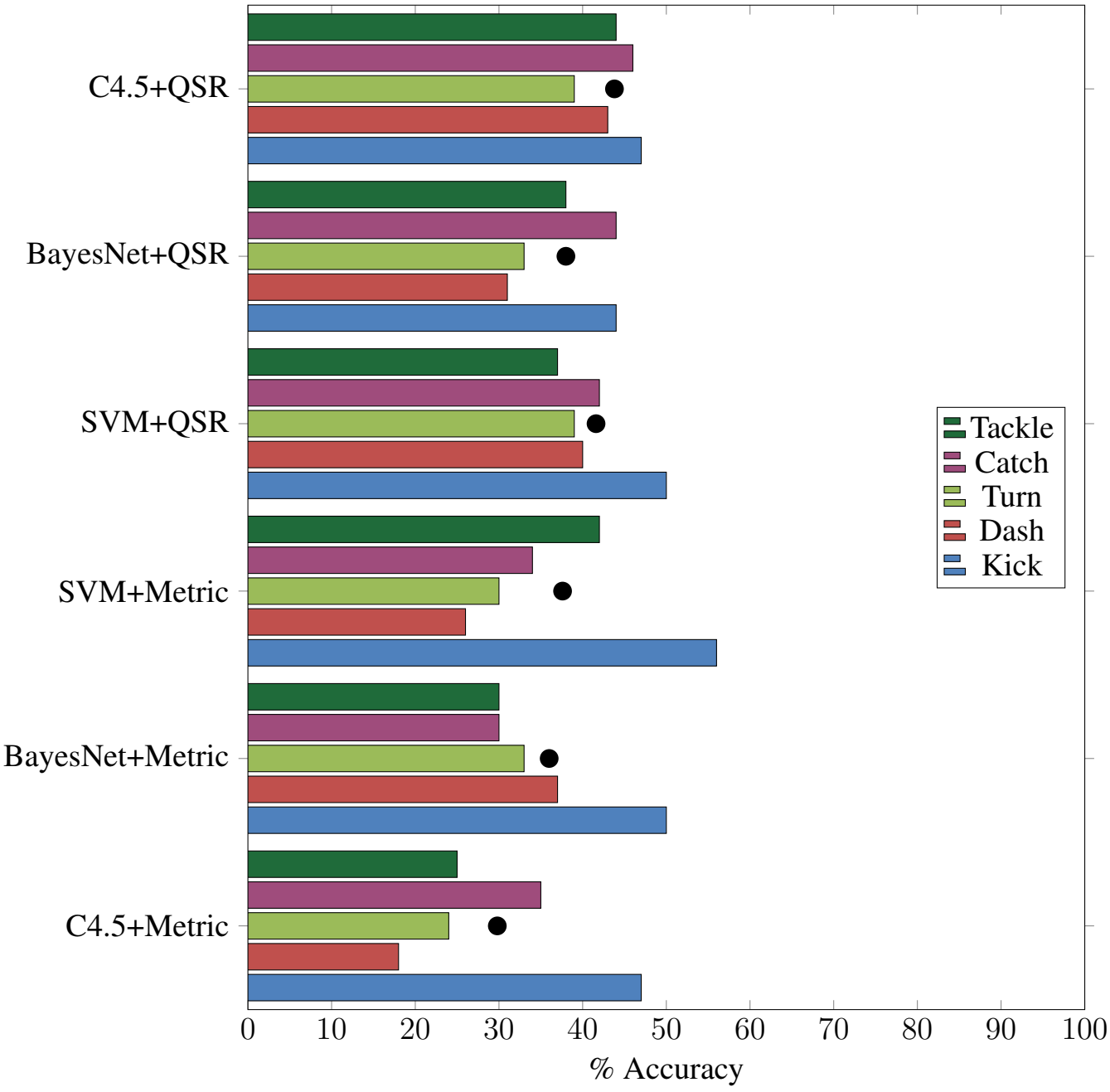
C4.5+QSR
BayesNet+QSR
SVM+QSR
SVM+Metric
BayesNet+Metric
C4.5+Metric
Tackle
Catch
Turn
Dash
Kick
0
10
20
30
40
50
60
70
80
90
100
% Accuracy

Figure 6.12:

| Name | Avg. Predictive Accuracy (C4.5) |
|---|---|
| AUA2D [99] | 78% |
| AUT [51] | 72% |
| Axiom [32] | 53% |
| Borregos [52] | 81% |
| CSU Yunlu [90] | 93% |
| DreamWing [98] | 43% |
| GDUT TiJi [36] | 95% |
| Helios [3] | 45% |
| LegenDary [72] | 81% |
| WrightEagle [8] | 39% |

Figure 6.13: Avg. predictive accuracy per team

The common characteristic for the majority of the results shows that the *Kick* and *Tackle* actions are often the most successfully predicted. This is likely because the situations where these actions are applicable are very rare – to kick the ball, the player must be near the ball, and to tackle a player they must be near a player that has the ball – these are discriminative factors, making them easier to predict. Also, it suggests that the states where this action is taken are very similar across most of the teams we examined. Whereas actions such as *Turn* and *Dash* are often executed in similar scenarios, making them slightly more difficult to predict. The results of all of our learning trials features this structure, but our system is best able to distinguish and extract the spatial characteristics of these situations. Even in the metric data this pattern is seen.

### 6.4.2 Deploying Models of Full-Game Agents

Deployment in the full-game scenario employs the same learning system as in the keepaway scenario. The only difference being that the system also recognises and is able to execute the additional *tackle* and *catch* actions, which are more commonly used in the full-game, specifically by goal-keepers. Since these appear naturally in the training data, the system requires no modification to detect, learn and utilise the actions itself.

To evaluate all teams we employed a round-robin approach where each team played one

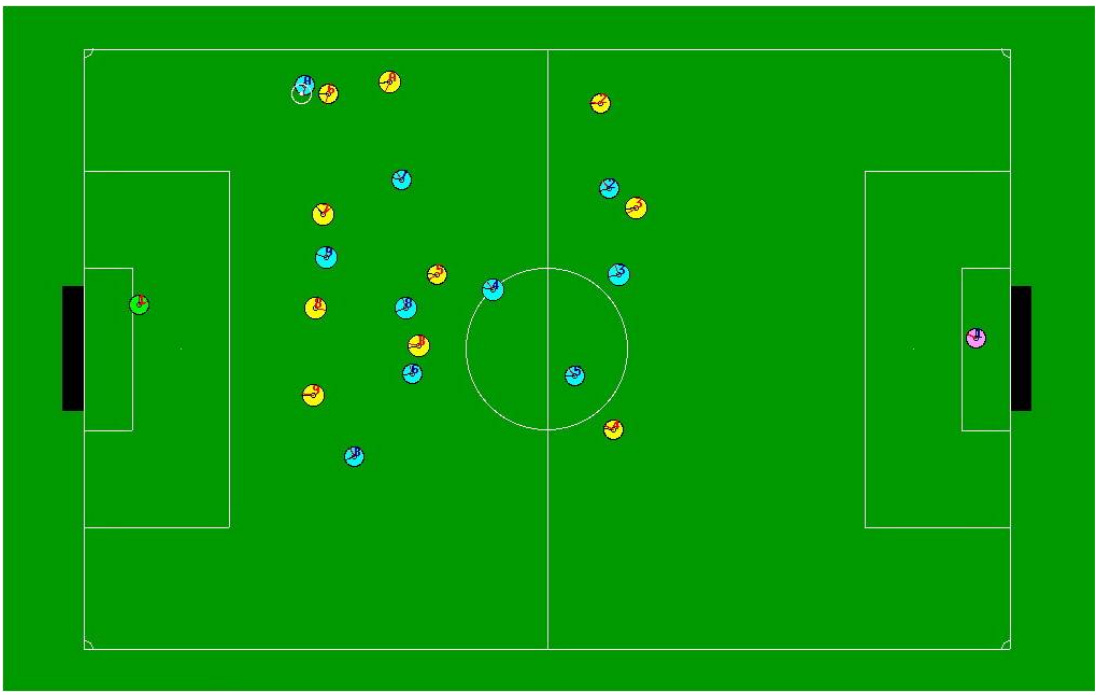

Figure 6.14: RoboCup Soccer – Full Game

| Opponent | Wins out of 100 (Original) | Wins out of 100 (Learned) |
|---|---|---|
| AUA2D [99] | 51 | 45 |
| AUT [51] | 51 | 46 |
| Axiom [32] | 44 | 36 |
| Borregos [52] | 85 | 78 |
| CSU Yunlu [90] | 65 | 59 |
| DreamWing [98] | 51 | 35 |
| GDUT TiJi [36] | 65 | 54 |
| Helios [3] | 95 | 27 |
| LegenDary [72] | 71 | 60 |
| WrightEagle [8] | 92 | 40 |

Figure 6.15: Results after round-robin experiment, shows wins for each team after playing 100 games per opponent.

| Opponent | Wins out of 100 (Original) | Wins out of 100 (Learned) |
|---|---|---|
| AUA2D [99] | 61 | 46 |
| AUT [51] | 57 | 43 |
| Axiom [32] | 44 | 31 |
| Borregos [52] | 82 | 72 |
| CSU Yunlu [90] | 45 | 39 |
| DreamWing [98] | 31 | 20 |
| GDUT TiJi [36] | 85 | 70 |
| Helios [3] | 25 | 12 |
| LegenDary [72] | 75 | 65 |
| WrightEagle [8] | 22 | 8 |

Figure 6.16:

hundred games against every other team. This provided a dataset from which LbO teams were produced, and the same experiment was run again where the LbO teams played one hundred games against each of the original teams. The results of this experiment are shown in Figure 6.15

An alternative way of providing training data to the system is to provide an isolated, per-opponent dataset, rather than combining all traces into a single dataset. In this way, the system learns a model for each specific opponent team, and then selects the appropriate model at the start of a game – by matching the opponent name against the set of known models. The results of training in this way are shown for CSU Yunlu in 6.17. The system exhibits improved performance, for instance increasing from an average win rate of 46% against the AUA2D team, to a 59% win rate, and increasing from 70% to 80% against GDUT TiJi. Training in this way provides around a 6% improvement in performance on average.

We can see that some LbO teams come very close to imitating the performance of their expert teams, however some fall far short, most notably the system taskes with imitating the Helios team. The split is largely between teams that use complex on-line learning or planning systems to dynamically generate behaviour, of which Helios is one, and those that use reactive planning and heuristic approaches, which the system is typically capable of detecting. We will

| Opponent | Win Percentage (Original) | Win Percentage (Learned) |
|---|---|---|
| AUA2D [99] | 61% | 59% |
| AUT [51] | 57% | 48% |
| Axiom [32] | 44% | 36% |
| Borregos [52] | 82% | 78% |
| CSU Yunlu [90] | 45% | 44% |
| DreamWing [98] | 31% | 25% |
| GDUT TiJi [36] | 85% | 80% |
| Helios [3] | 25% | 17% |
| LegenDary [72] | 75% | 70% |
| WrightEagle [8] | 22% | 10% |

Figure 6.17: Performance for the CSU Yunlu team, the base performance of the team is given, averaged over 10 games per opponent, as well as the performance of a team of learning agents using models based on CSU Yunlu, though trained per-opponent, also averaged over 10 games per opponent.

look at this more in the analysis chapter.

## 6.5 Experiments with Starcraft

## 6.6 Tasks

We selected three benchmark scenarios of increasing difficulty from an online repository of Starcraft micromanagement training maps [1] intended to allow Human players to practice specific micro-management skills. These include a kiting task, a group micro-management task, and a final, more difficult task involving a sequence of group combat scenarios of increasing difficulty and opponent make-up. These were introduced in chapter 5, but we will briefly recap them here for convenience.

[1] http://wiki.teamliquid.net/starcraft/Micro_Training_Maps

Figure 6.18: An example of a Kiting scenario in Starcraft

**Task One – Kiting Task**

In this scenario a single fast-moving ranged unit is faced with constantly spawning waves of fast-moving melee units and must kite increasingly large groups, while scoring points for destroying pursuing enemies. Success is based on the ability of the controller to keep the number of enemies under control, while continuously moving out of danger. Attacking will cause the unit to momentarily stop, allowing the enemies to gain ground. There is no time limit on this task, and it continues until the vulture is inevitably destroyed, which is guaranteed to happen due to the rate at which enemies appear. Destroying an enemy unit earns a point, and the performance measure we use is the final score when the vulture is eventually overcome. A video of a benchmark agent performing the task is available online [1].

**Task Two – 3vs2 Combat**

In our second task, we repeat the experiment of [60]. In this task, the player is given control of two Dragoons (large, ranged units) and is tasked with defeating a squad of three Hydralisks from the Zerg race (medium-sized ranged units). The player must make aggressive use of the advantages of their own units by splitting their force up, and ensuring that the enemy units cannot focus fire on a single unit. Kiting tactics could be used, but this may be difficult for a Human player since it would require controlling multiple units moving in different directions simultaneously. The initial scenario setup is shown in Figure 6.19. In this case a win or loss

[1] https://vimeo.com/103123818

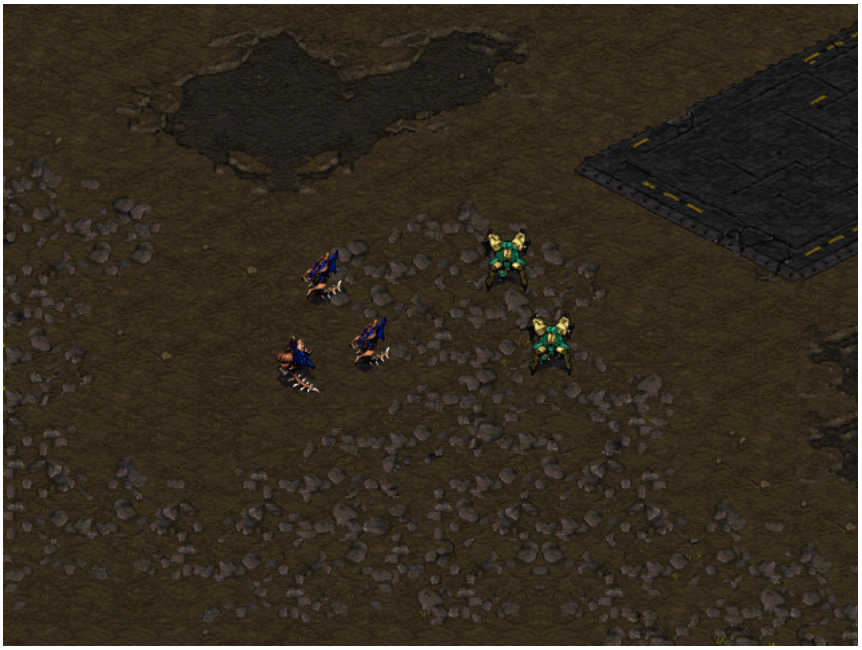

Figure 6.19: The 3vs2 scenario which is the focus of the experiment of Parra[60]. The Dragoons (in teal) move faster than the Hydralisks (in blue), but the Hydralisks deal far more damage to the Dragoons than vice-versa, and also outnumber them, meaning that a focused attack from all three would destroy a Dragoon almost instantly.

is marked by destroying all of the opposing units, or having your own units destroyed, with no time limit.

**Task Three – Group Combat Sequence**

In the final scenario, we consider a more difficult task – a scenario named *The Fall of the Empire*. Initially a player is given access to four Dragoon units, and must progress around a series of corridors, defeating groups of enemies along the way. The player is

### 6.6.1 Learning Models

The learning framework for the Starcraft domain is nearly identical to that used in the RoboCup domain. The only difference being in the information used to calculate $QTC_b$ symbols – specifically the *"... is moving faster or slower"* symbol. In RoboCup, this is based on the relative movement velocities of players. In Starcraft, there is no concept of velocity or acceleration, as all units move at a constant speed defined by their type. As such, in Starcraft we generate this

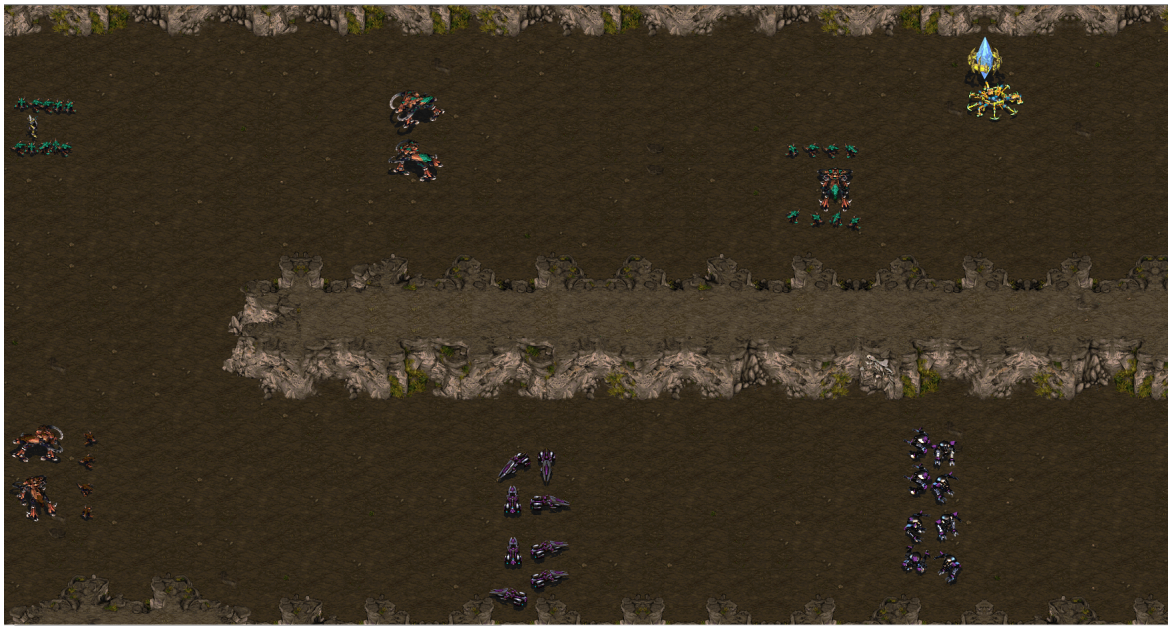

Figure 6.20: Two corridoors along which the agent's army must move in the *The Fall of the Empire* scenario

symbol based on which unit has a faster base speed than the other, and whether the units are in motion or not. Two units in motion that have identical speeds will be declared as stable in $QTC_b$ terms.

### 6.6.2 Deployment in Starcraft

The system to control agents in the Starcraft domain differs slightly from the one used in RoboCup, in RoboCup to achieve locomotion an agent will first turn to a desired angle using the turn action, and apply an impulse using the dash action. In Starcraft, an agent instead directly chooses a *point in space* to move to, which takes the form of numerical parameters passed to the movement action. In RoboCup we needed to construct a deployment mechanism to recover the numerical parameters of actions, and deployment in Starcraft requires the same kind of system. Since a movement action in Starcraft describes a point in space, the system translates this into the QSRs of that point relative to the agent that performed the action, as well as any environmental landmarks such as invariant spatial regions, enemies, allies etc. As such, the action descriptor specifies a point that possesses certain relational properties with regards to the environment, which may in fact be shared by a range of points in the metric space.

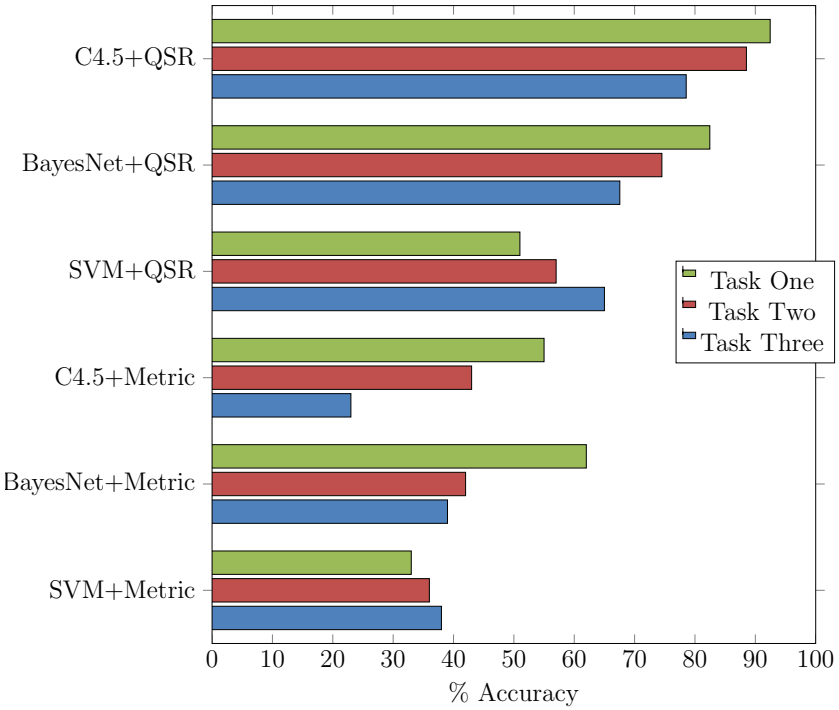


Figure 6.21: Per-task performance of classifiers in the three Starcraft benchmark tasks described previously.

When we come to deploy a model, we *decompress* the QSR information by sampling the metric space around the agent and selecting points that most closely match the QSRs described by the action. This is accomplished using the Monte-Carlo sampling approach discussed in previous chapters.

We also utilised this approach for the targetting system. In the Starcraft data when one unit attacks another the action is is specified in the form *attack[unit ID]*. This is not exactly what we would like, as when we come to attempt to re-use that action in the future the unit with a specific ID may not exist. Instead, similar to how we record movement actions, we record the QSRs of the unit that is the subject of the attack action. When the system comes to execute that action in the future, it then seeks out the unit on the field that most closely matches the QSR description in the action. This is similar to the way a human would choose to attack a unit – by issuing an attack command and clicking on an enemy.

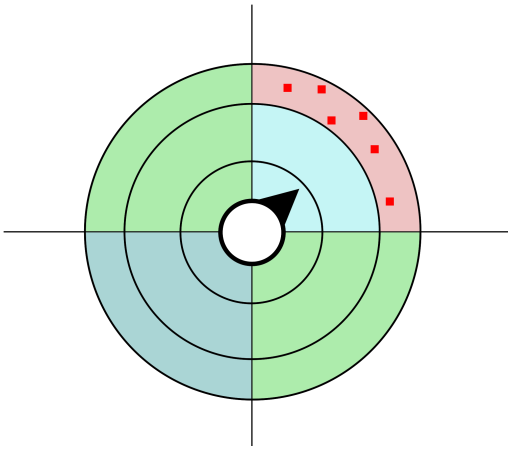

Figure 6.22: Sampling points based on QSRs. Many points in metric space – the red dots – may fit those specified by an action.

### 6.6.3 Deployment Results

In each experiment we provided the learning agents with full data sets (150 episodes per task), and evaluated them for 100 episodes. Figure 6.23 and Figure 6.26 shows our results for the Vulture kiting scenario, giving the average score of each system in terms of number of kills, which is the performance metric provided by the scenario. There is a slightly larger drop in performance when the system is trained on the Human data than synthetic (). This deficit can be explained by the fact that the Human controllers were users who had previously limited experience with Starcraft prior to the study. As such, while the synthetic agents achieved a roughly consistent level of performance across all episodes, some Human data included initial episodes where the Human controller learned the task, and performed relatively poorly during those episodes. This allowed mistakes and errors introduced by the expert to propagate to the learning agent. This shows as the slope in Figure 6.25, which illustrates the average score per episode for all Human controllers and benchmark agents.

Again, as in the RoboCup experiments, our approach was to combine all learning traces of the Human data in to a single training set, and this comprises our results in Figure 6.23. Alternatively we can train individual models from each Human. One instance of this is shown in Figure 6.23 where we have selected and trained an agent based only on the best performing Human in the dataset. Figure 6.24 shows an expansion of this experiment, with the raw perfor-

| System | Avg. Score |
|---|---|
| Human Controller (Single Best) | **22 ± 1.13** |
| Benchmark Agents | 20 ± 3.14 |
| Learned (From Best Human) | 19 ± 2.41 |
| Learned (From Benchmark) | 17 ± 3.88 |
| Human Controllers (Average) | 17 ± 3.96 |
| Learned (From All Humans) | 14 ± 4.47 |

Figure 6.23: System Results for Vulture Kiting Task, given in terms of average score over 100 episodes, *including* the initial learning curve data.

| System | Avg. Score |
|---|---|
| Human Controller 4 | **22 ± 1.13** |
| Human Controller 3 | 20 ± 1.04 |
| Learned (From Human 4) | 19 ± 2.41 |
| Human Controller 5 | 17 ± 2.44 |
| Learned (From Human 3) | 17 ± 1.83 |
| Learned (From Human 5) | 15 ± 2.05 |
| Human Controller 1 | 15 ± 2.08 |
| Learned (From Human 1) | 11 ± 4.33 |
| Human Controller 2 | 12 ± 4.14 |
| Learned (From Human 2) | 8 ± 1.13 |

Figure 6.24: Performance of Human subjects in the Vulture Kiting task, alongside agents trained on the just a single particular Human.

mance data of each of our Human subjects alongside performance data of agents trained only on each Human.

In the experiments of Parra [60] it was observed that the built-in AI of Starcraft was unable to achieve victory at all, whereas their Bayesian Network-based system trained on the observation of Human micromanagement achieved victory in 44% of cases. These results, along with our own, are displayed in Figure 6.27, Parra does not cite the underlying win rate of the observed Human experts in their experiment, but in our repetition of this experiment this was 67%, similarly in our repetition the built-in AI achieved a win rate of 0%. The results show that our benchmark agents perform well on the task, as do agents that have learned to imitate them. Human controllers however achieve a much lower level of success, as do the imitating

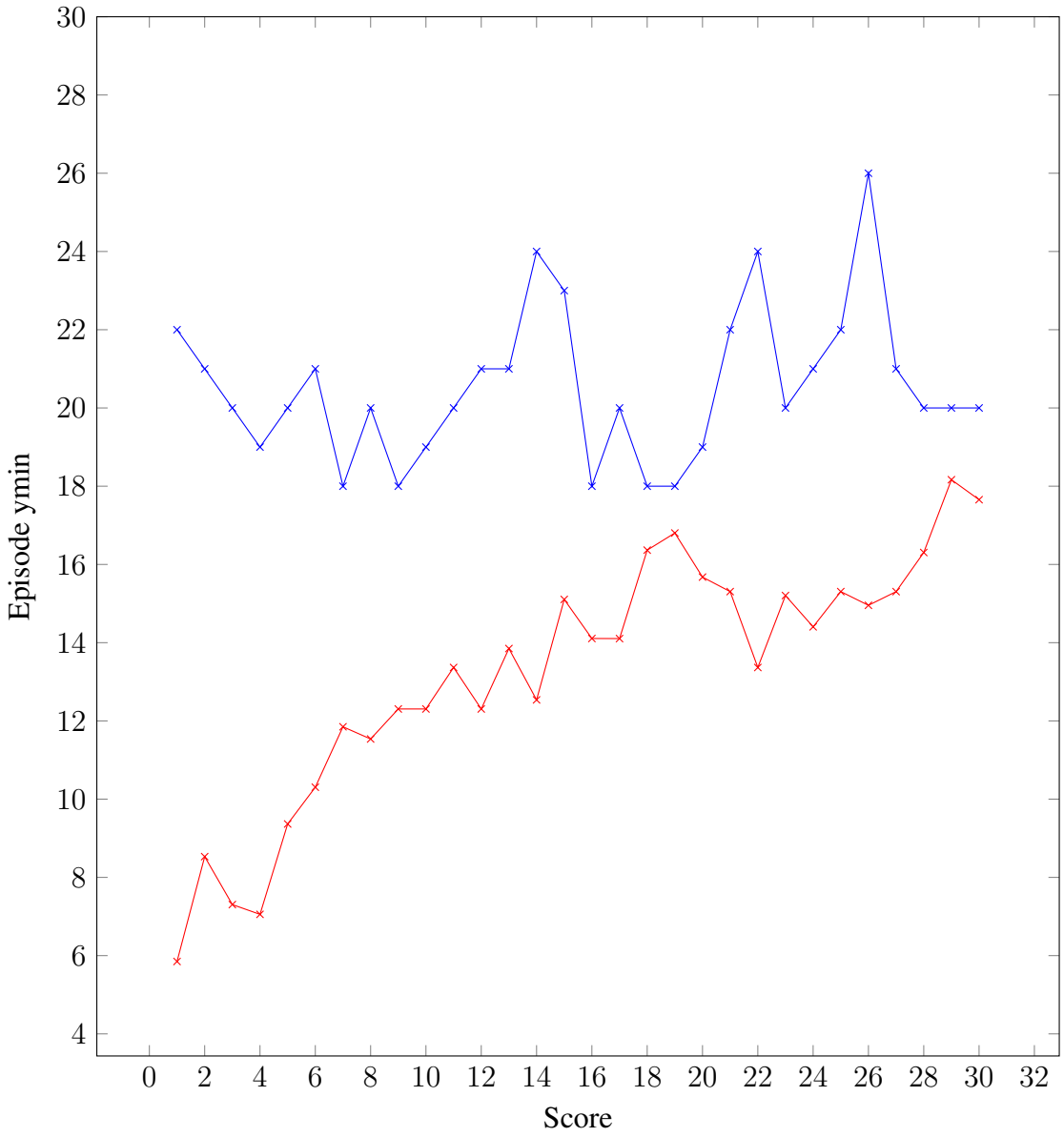


Figure 6.25: Average scores in the kiting task for all Human test subjects (red) and benchmark agents (blue), showing that Human controllers exhibit a learning curve, whereas performance of synthetic agents falls within a static range

| System | Avg. Score |
|---|---|
| Benchmark Agents | **20 ± 3.14** |
| Learned (From Benchmark) | 17 ± 3.88 |
| Human Controller | 16 ± 4.43 |
| Learned (From Humans, best humans) | 15 ± 2.15 |
| Learned (From Humans, all humans) | 11 ± 3.37 |

Figure 6.26: System Results for Vulture Kiting Task, given in terms of average score over 100 episodes, *excluding* and *including* the initial learning curve data.

| System | Avg. Win Rate |
|---|---|
| Benchmark Agents | **96**% |
| Learned (From Benchmark) | 92% |
| Human Controller | 67% |
| Learned (From Humans) | 58% |
| Learned (Parra [60]) | 44% |
| Built-in AI | 0% |

Figure 6.27: System Results for 3vs2 task, given in terms of average win rate over 100 episodes.

| System | Avg. Score |
|---|---|
| Benchmark Agents | $\mathbf{42 \pm 7.43}$ |
| Learned (From Benchmark) | $32 \pm 5.13$ |
| Human Controller | $19 \pm 8.11$ |
| Learned (From Humans) | $16 \pm 6.24$ |

Figure 6.28: System Results for The Falls of the Empire, given in terms of average score over 30 episodes.

agents. This is because the task is very difficult – the best strategy is to split one's forces off moving in multiple directions, and engage in a kiting behaviour similar to the first task, and this is what our synthetic benchmark agents do. But, this requires monitoring multiple units in different places on the map – this is very difficult for a Human to do, but very easy for an AI to do. The Human players instead preferred to kite the opposing units around the area where the task begins, keeping them visible at all times. This is a far more dangerous and less effective strategy, but significantly easier to execute for a Human. The resulting LbO agents trained on the Human behaviour therefore behave accordingly, executing a strategy that is generally poor.

The *Falls of the Empire* task proved very difficult for the Human players, with very few progressing past the first stage of the map, which features five separate combat encounters of increasing difficulty, against 24 total groups of enemies. The benchmark agents typically progressed in to the second stage of the map, but rarely further than this, which illustrates the difficulty of the task. This made training models particularly difficult, and as Figure 6.28 shows, the system is capable of learning some of the behaviours required to succeed at the task. The Human and synthetic controllers both made use of kiting tactics similar to those employed in the previous experiment, which the learning system was able to imitate very closely, to some success.

| System | Avg. Score |
|---|---|
| Human Controller | $25 \pm 5.51$ |
| Learned (From Humans) | $20 \pm \pm 3.84$ |

Figure 6.29: System Results for The Falls of the Empire, given in terms of average number of kills over 30 episodes, when removing the two worst Human players from the data set.

Our results show that our approach is successful in learning low-level tactical skills, and achieving a level of performance approaching that of the original expert controller. However we also see that there is a clear coupling between the performance of an expert and the agent which imitates it, including the replication of mistakes. This is seen most starkly with the Human controllers. In the third task, the two worst Human players managed, over 30 episodes, average scores of 3 and 7. Expert players in this task average scores of around *50-60*, and so our average Human controller score of *19* is generally poor. When we remove the traces of the worst Humans from the data set, and train the model on the three remaining Humans, the results are slightly improved, as shown in Figure 6.29.

One thing we have not touched on yet is that Starcraft also has a wide range of meta data about the environment that is not added to our representation. While spatial information alone seems to go quite a long way in these tasks, in some cases it does not provide all of the information that is needed to make a good prediction. For instance, a unit in Starcraft might have an instruction to always attack the nearest enemy with the lowest amount of health. For this, spatial information makes up only 50% of the information used by the agent to pick its next action.

We actually do know that this kind of logic is used by the benchmark agents – since they were developed by us for an unrelated project. In small battles, such as the first and second task, this information is not hugely beneficial, but in larger battles as seen in the third Starcraft task it can be crucial. It would be trivial to simply include the metric value of each unit's health to our state descriptor – as a normalised value from 0% to 100% – but that would bring in a lot of the problems and difficulties associated with metric values we have discussed previously.

Instead, we utilised the ring calculus again to break down the health bar of a unit into discrete intervals, giving us a *qualitative representation of unit health*. This is then added as

| Health Value (%) | Symbol Score |
|---|---|
| $0 - 19$ | Low |
| $20 - 39$ | Medium-Low |
| $40 - 59$ | Medium |
| $60 - 79$ | High-Medium |
| $> 80$ | High-Medium |

Figure 6.30: Lookup table for our qualitative representation of unit health

| System | Avg. Score |
|---|---|
| Benchmark Agents | **42 ± 7.43** |
| Learned (From Benchmark) | 38 ± 5.13 |
| Human Controller | 25 ± 5.51 |
| Learned (From Humans) | 23 ± ± 2.14 |

Figure 6.31: System Results for The Falls of the Empire, given in terms of average number of kills over 30 episodes, when we include the qualitative representation of health.

a non-relational value in our state description, the lookup table is shown in Figure 6.30. This not only allows the system to include information about the health of enemy units, but also its own units, so instructions such as "Run away when you are low on health" can be more easily captured.

Rerunning our experiment with the benchmark agents and agents trained on their traces gives us the slightly improved results in Figure 6.31. We engineered the thresholds for the representation of health using our own knowledge of domain, and excluded it from the process of representation configuration. The parameters certainly could be subject to the configuration process, to capture actions where metric information is an important deciding factor.

## 6.7 Additional Experiments on training data requirements

An additional factor we wished to investigate was the amount of training data required by our system to achieve a given level of performance. The compressive nature of QSRs means that many different metric instances can be encompassed within a single QSR instance, this improves the number of repeated states in the data and makes the learning task easier. Analysis of our RoboCup Soccer Keepaway 3vs2 dataset, after the representation selection process, reveals that 53% of states appeared more than once, with 41% of the total appearing more than three

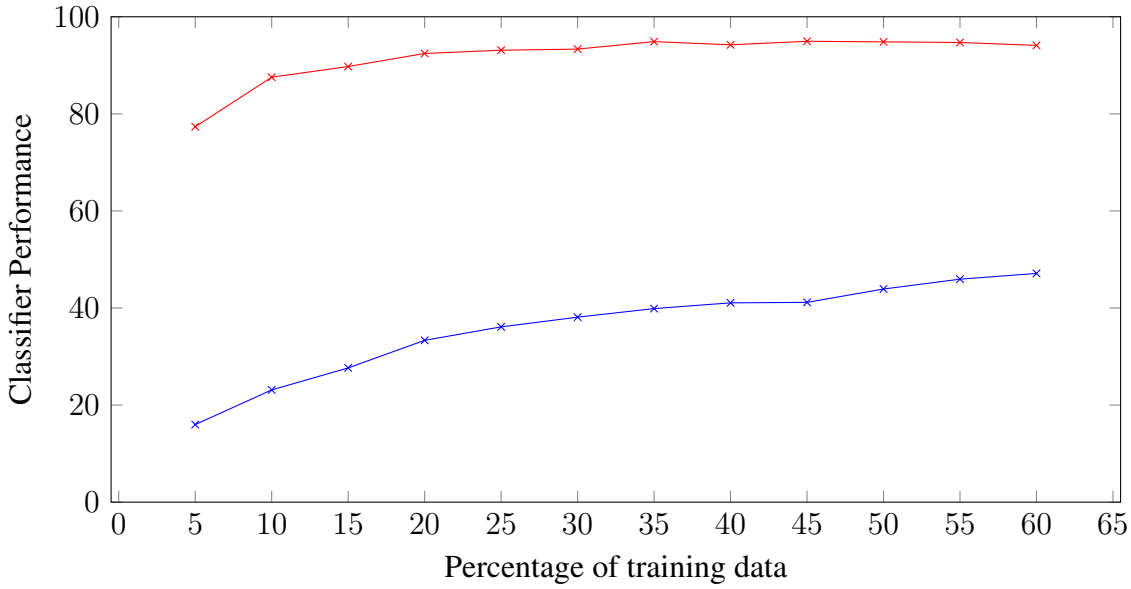


Figure 6.32: Training data needs analysis in the Starcraft Kiting scenario using benchmark agents. Blue line reports the performance of the C4.5 classifier using a metric representation, and the red line reports the performance of the C4.5 classifier using a QSR representation, with representation selection.

times. In contrast, the same analysis of the dataset for the more complex full-game scenario reveals that only 27% of states appeared more than once, with 13% of the total appearing more than three times.

This causes us to ask the question: How much does a system need to observe in order to perform well? All learning mechanisms are subject to the law of diminishing returns. As the learning process gradually exhausts the amount of new information that can be extracted from the training data, eventually the only things left to learn are rare edge-cases, which provide limited benefit.

To investigate the question of training data requirements in our own system, we ran experiments where we varied the amount of training data provided to our system. A graph of this is shown in Figure 6.33.

The vertical axis indicates the average performance of a system over ten-fold cross-validation, the horizontal axis indicates the percentage of the training data provided to that system. We selected training data randomly on a per-episode level of granularity, so agents are provided with randomly selected complete episodes rather than randomly selected individual *frames* of data

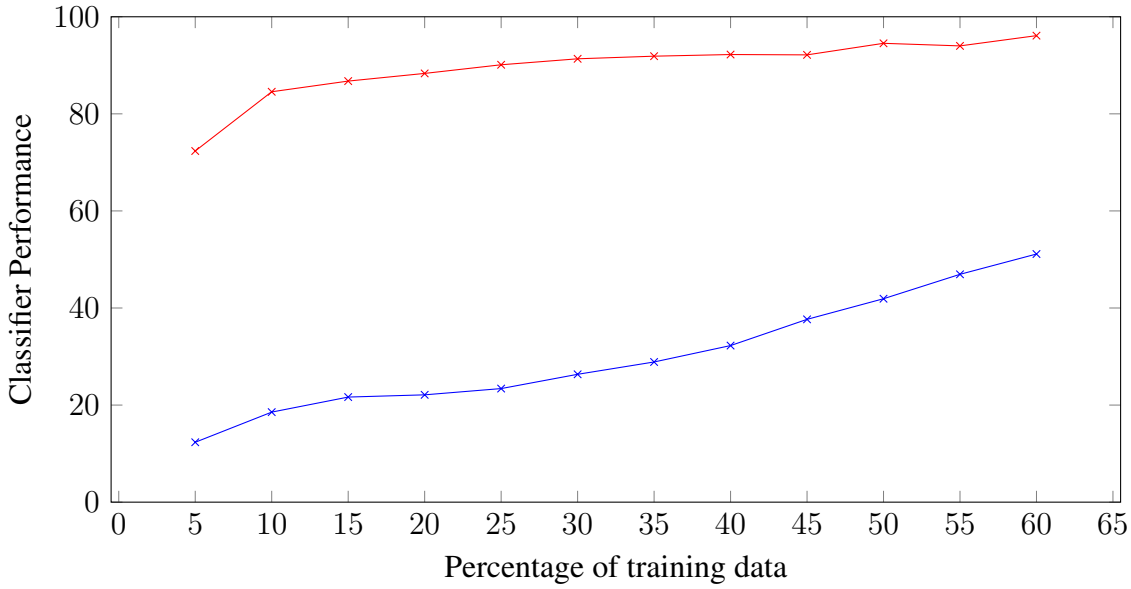


Figure 6.33: Training data needs analysis in the RoboCup Soccer Keepaway scenario using benchmark agents. Blue line reports the performance of the C4.5 classifier using a metric representation, and the red line reports the performance of the C4.5 classifier using a QSR representation, with representation selection.

taken from episodes. For instance, our data set for the 3vs2 task contains 1000 episodes, 3% of this (the third data point on the graph) would equate to 50 random episodes taken from the data.

The average performance of the QSR based system in our previous experiments was 96%, with 100% of the training data provided to it. Here the system gets very close to this performance level with a fraction of the training data. With 50% of the training data (500 episodes) the average performance of the system is 94.53%. The system using a metric representation, which converges to an average performance of 74% is still improving at this stage. This shows that our system learns the task very quickly from a small sample of examples. We repeated this experiment in the Starcraft kiting task, the results of which are shown in Figure 6.34, and exhibit similar characteristics to the RoboCup experiment. In both cases both systems reach a level of performance within 5% of their eventual peak performance after observing around 20% of the provided training data in these cases.

In Figure 6.35 we show the results of deploying QSR-based agents given various portions of the training data from 1% to 100%. The system roughly reaches its peak performance after observing only 25% of the training data provided, equating to 250 episodes, but the system is

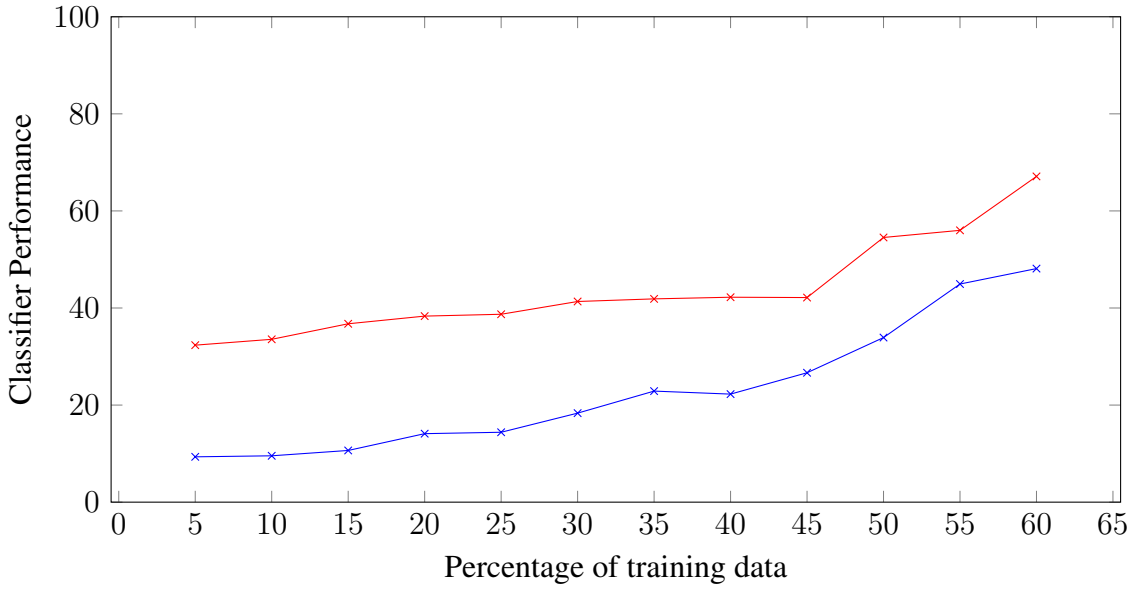


Figure 6.34: Training data needs analysis in the RoboCup Soccer full-game scenario using the CSU Yunlu agents. Blue line reports the performance of the C4.5 classifier using a metric representation, and the red line reports the performance of the C4.5 classifier using a QSR representation, with representation selection.

close to this after observing only a randomly selected 10% of the training data. This can be further improved upon by selecting training instances in a more principled way. For instance, as we have observed that the training data contains many repeated states, we can turn this into a metric with which to evaluate the estimated *value* of a particular episode, in terms of the proportion of new states to states the system has already seen that it contains. Starting with a random training episode, we then add episodes to the training set based on the proportion of new episodes to old, preferring those that have more unique states, until the training set for the system encompasses some pre-set portion of the whole – for this experiment we chose 5%. The result is shown in Figure 6.35, and shows that the system with training data selected in this way slightly outperforms systems provided larger amounts of randomly selected training data. This has other practical benefits, for instance minimising the training data in this way reduces the time taken for learning – we are essentially trading (expensive, slow) learning steps for (inexpensive, fast) pre-processing steps.

A deeper analysis of training data characteristics is certainly possible. Here we have analysed the value of states on an instance-by-instance level, but we expect there to be cases where

| Training Data | Avg. Episode Length |
|---|---|
| 100% | $123 \pm 6$ |
| 50% | $122 \pm 6$ |
| 5% (ranked) | $119 \pm 4$ |
| 25% | $121 \pm 7$ |
| 10% | $117 \pm 7$ |
| 3% | $111 \pm 9$ |
| 1% | $107 \pm 12$ |

Figure 6.35: Average episode duration in simulator steps showing variation in performance after learning from both Human and benchmark agents, with the amount of training data varied per system, and including the system with training data select that is subject to our ranking algorithm.

the *sequence* of states is important too, where two or more states may not have been seen before, but where their sequential co-occurance may not have been observed, and thus might provide important information to the system. There are trade-offs with such deeper analysis however. Pre-processing will always add some degree of overhead to the performance of a system, and discovery of arbitrary-length sequences is a good example of what can be a heavy operation, but restricting that pre-processing to simple operators, as we have done here, can be prove beneficial with minimal overhead.

# Accelerating Reinforcement Learning with Learning by Observation

So far we have mainly been concerned with the task of imitation, with our agents attempting to copy what they observe as closely as possible. One problem with this is that since their learning is unsupervised in the sense that the agents are not provided any measure or information about which actions and behaviours are *beneficial* within the context of a domain, they copy *detrimental* actions and mistakes just as they would *beneficial* actions. We experimented with several ways to alleviate this problem, with the most successful being to statistically analyse the training data and, making the assumption that experts will make mistakes rarely, remove actions that, in specific states, differ significantly from the actions that would be taken on average. The problem with doing this kind of data cleaning is that it may remove the agent's ability to respond to rare, exceptional situations that may indeed *warrant* sets of actions that differ from the norm. In RoboCup we found that the full-game scenario produced these kinds of situations far more than the KeepAway scenario, likely due to the complexity of the state space. Nevertheless this is a complex and interesting problem. Our results show that while the performance of our LbO agents gets close to the performance of the original agents (and especially when using our self-configured QSR approach) it typically fails to exceed or even exactly match the performance of the underlying agent. There is ultimately a theoretical limit on the performance of an LbO agent using our approach. If we were to attain a perfect imitation of the expert agent, the task-level performance of our agents would still be limited by the task-level performance of the expert. As such, if we chose a poor expert to observe, the resulting agent would in turn behave

poorly. This can be thought of as the difference between *imitation performance* and *application performance*.

This chapter presents further experimentation by using Reinforcement Learning (RL) on top of our Learning by Observation framework, producing a novel extension to RL. We re-use both our learning and model deployment frameworks, and combine our LbO agents with RL techniques to produce an agent that inherently skips much of the initial exploration required by pure RL agents. In the RL component we utilise standardised reward functions from the literature.

Further work in the RL community also relates to our own, especially the use of *tile coding* as a form of function approximation [77, 76]. RL algorithms also must deal with metric parametrisations of state-spaces, and so tile codings are often used to define abstractions over that space, and these have been applied to produce representation abstractions similar to the QSRs that we make use of, though are usually not relational. Work also exists on learning effective tile codings for particular functions [93], for instance to provide abstractions over the parameter space for actions.

The work of Griffith [35] has similarities with what we do here. Here, the input of human feedback is used in order to shape the action selection policy of a RL algorithm, evaluated in the domains of Frogger and Pac Man. This allows a human to provide feedback to a system regarding the suitability of an action in a particular state. A similar approach is taken in the work of Thomaz [85], where again human teachers are included in-the-loop of an RL algorithm. The work of Price [62] a domain expert in the loop, but this takes the form of an oracle which can be communicated with by the learner. We found no work relating our specific intended use case – learning a predictive model from limited observations of a domain expert, and using this in the RL loop.

**Inputs:** A set of states S, a set of actions A, an action selection policy $\varepsilon$
**while** *!terminated* **do**
    Choose an action **a** from A according to the policy $\varepsilon$
    Observe reward **r** and new state **s'**
    Choose an action **a'** in **s'** according to the policy $\varepsilon$
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)$
    $s \leftarrow s'$
**end**

**Algorithm 4:** The SARSA Algorithm for Q-learning

## 7.1 SARSA

In our RL work we make use of the SARSA algorithm [44] which, while a minor variation on the Q-learning algorithm, typically yields better results than the base Q-Learning algorithm. Typical Q-learning seeks to approximate an action-reward function through exploration of the action space, updating the Q-value of a state-action pair based on a learning rate and discount factor. The algorithm works by executing an action according to some action selection heuristic and recording the reward that is received. The SARSA modification instead selects a second action after the first, in the new state, and bases the Q-value on this. In short, SARSA bases its Q-values on the reward gained from an action that is taken in a state, as well as the reward from the action that is taken in the subsequent state. This allows for more exploration of the state-space and discovery of actions that produce good results when executed consecutively. SARSA is a widely used standard in the RL community [40, 71, 48, 100] and has been employed in RoboCup [48] and Real-Time Strategy games like Starcraft [69] – as well as to teach systems to play Super Mario [54, 42]

### 7.1.1 Reward Functions

The primary issue with using Reinforcement Learning is the need to design a reward function. This can prove difficult depending on the characteristics of a given domain, and the principles of designing good reward functions is a field of study of its own. Robocup Soccer Keepaway is partly an interesting research platform because, while the behaviours of agents can be complex, the mechanics of the game mean that reward functions can potentially remain quite compact.

The goal of the Keeper team is ultimately to maximise the length of the episode, and so it makes sense to provide reward to those agents proportional to episode length. This is the same reward function as used by [83]. Similarly, the Starcraft Vulture Kiting task can be seen as having a very similar reward function – the agent must keep the unit alive for as long as possible. One difference however is there is an explicit scoring mechanism – the number of enemies killed – and this is in fact what we base our assessment of the agent's performance on. In the case of this task, we include a reward for killing an enemy as well as a small continuous reward for staying alive.

## 7.2 Action Selection

Reinforcement learning algorithms are typically used when a system has no prior knowledge of the action-reward space, and so must explore that space to build up a model, typically using heuristic estimates of expected reward in order to guide action selection. It is the action selection policy that we make modifications to in order to integrate our Learning By Observation system. Two common action selection policies for Q-learning are as follows:

> $\varepsilon$**-greedy** selects the action with the highest estimated reward with probability $1-\psi$ where $\psi$ is a probability of uniformly selecting an action at random.
>
> **softmax** ranks actions based on their estimated action-value, and non-uniformly selects actions based on a weighting derived from this ranking.

These are by no means the only action selection policies used in the literature, and there exists a wide range of work on designing and even learning good action selection policies [39, 92, 86]. As mentioned previously, our integration of LbO and RL takes the form of a modification to the action selection policy used by a SARSA algorithm. Our approach is algorithm-agnostic however, and any RL algorithm that makes use of an action selection policy could be employed in place of SARSA. We begin by training a predictive model on the expert data as detailed in the previous chapter. Our new action selection policy then functions as follows:

**EXPERT-LbO** With probability $\sigma$ select the *softmax* method, and thereby rank the set of known actions based on their estimated reward, select accordingly. With probability $\varphi$ (where $\sigma + \varphi = 1$) select the action predicted by the LbO system in this state. If this action is selected, with probability $\rho$ apply a Gaussian noise to the action parameters, if it has any. With probability $1 - \rho$ execute the action unmodified. Execute the action and return it to the set of known actions.

The underlying hypothesis of this work is that it may be beneficial for the algorithm to start its search outwards from some known-good states, which are provided by the expert data. Providing peices of domain information to an algorithm in order to jump-start its initial performance is sometimes called *bootstrapping*. While the system does not initially know the state-action-value mapping, the expert data provides a set of possible actions that can be considered as good starting points that we expect to be better than random trial-and-error starting conditions. The ability of the system to *modify* those actions is based the observation that experts may not themselves be perfect actors, and that their behaviours may be able to be improved upon. Eventually, with enough trials, every action taken by the expert will be evaluated by the RL algorithm. This further works to filter out mistakes and actions resulting in poor performance, as those actions will yield poor rewards, which is a value judgement we have previously not been able to make.

## 7.3 Algorithm

As mentioned, our base RL algorithm is the Q-learning algorithm SARSA shown in algorithm 7.1, with our sole alteration being to the action-selection policy, which we named **EXPERT-LbO**.

The parameters $\sigma$ and $\varphi$ affect the system's performance greatly, and must be tuned somehow. To do this we apply an idea from the Simulated Annealing algorithm [14] which applies what is commonly known as an *acceptance rate* which changes over time, in Simulated Anneal-

ing this is typically the probability of the algorithm selecting a radically new, different solution in an attempt to explore more of the search space, but we make use of it slightly differently. In this work, we use this acceptance rate to set the values of $\sigma$ and $\varphi$ so that the likelyhood of selecting the expert's action is initially high, but will decrease over time, $\sigma$ thereby increases and thus the system tends towards preferring the *softmax* algorithm. This gives us a further parameter $\gamma$, which determines the way the acceptance rate changes over time. The simplest value for this would be a constant, so as to produce a linear decrease in the acceptance rate over time, but non-linear functions can also be used, so as to modulate between phases of exploration and exploitation over time.

## 7.4 Experiments

We selected two experiments from each of our test domains to repeat with the addition of RL, these being the 3vs2 RoboCup Keepaway task and the Vulture Kiting experiment. The Vulture Kiting task was chosen as it theoretically has no end – a given episode can continue on for as long as the controllers are capable of continuing, whereas the other Starcraft tasks we evaluated had clear end-states where the scenario finishes when all enemy units are destroyed. This simplifies the engineering of the reward function a great deal.

We compare three different systems.

1. A RL system utilising a metric-based representation, labelled RL-METRIC, learning actions by its own trial-and-error.

2. A RL system utilising the QSR-based representation, but learning actions by its own trial-and-error, labelled RL-QSR

3. A RL system utilising the QSR-based representation and action selection criteria discussed so far in this chapter, labelled RL-QSR-EXPERT

### 7.4.1 RoboCup Soccer

We began by training our LbO system using the training data ranking technique described in the previous chapter, selecting only the best 5% of the training data. From there we allowed the RL systems 1000 trials, utilising the algorithms detailed previously. Our results are shown in Figure 7.1

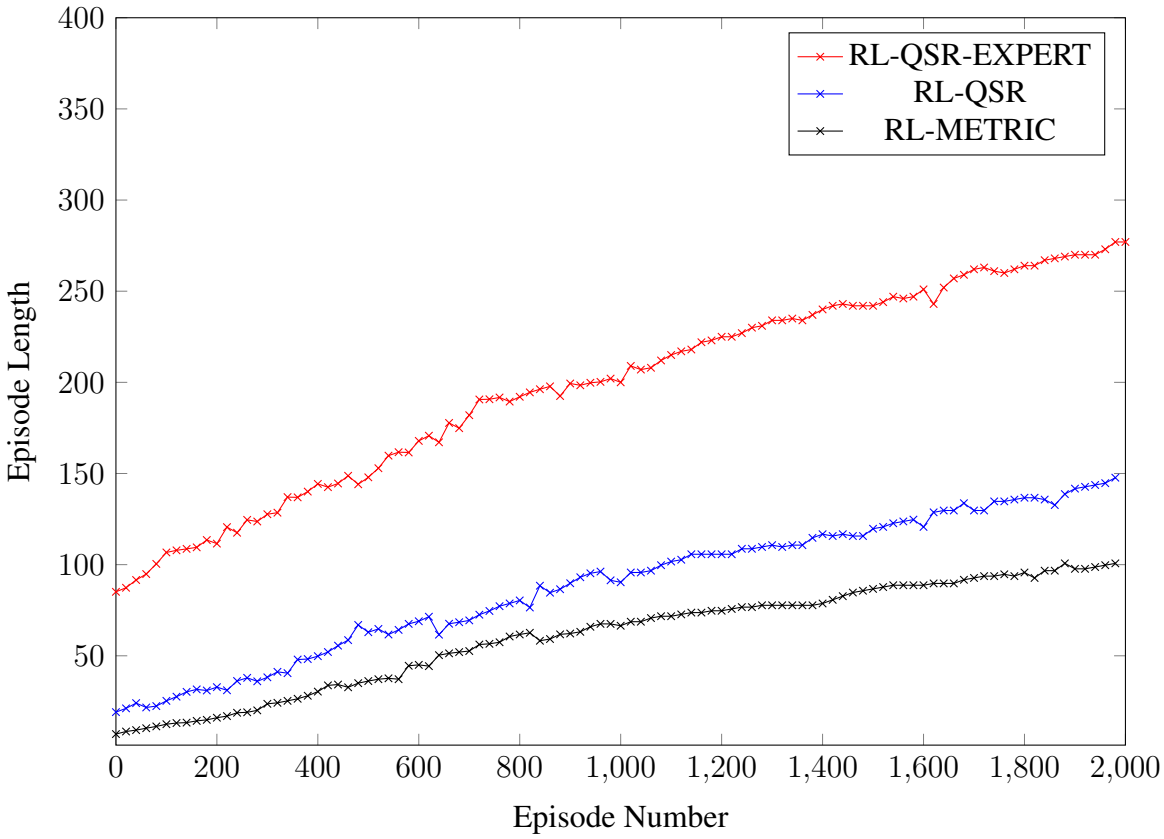


Figure 7.1: Results of Reinforcement Learning experiments in RoboCup Soccer keepaway, 3vs2 configuration.

The results in Figure 7.1 are shown in intervals of 20 episodes, with there being 19 episodes between each data point. The results show that the RL-QSR-EXPERT algorithm significantly outperforms the other two algorithms, eventually reaching episodes of **201** time steps, which is better performance than the underlying expert agent which performed at an average of 151, and also better than the best single human controller which averaged 186. The RL-QSR algorithm eventually reaches around 96, and the RL-METRIC algorithm reaches 63. The RL-QSR-

EXPERT algorithm was configured with initial values of $\sigma$ and $\varphi$ at $0.4$ and $0.6$ respectively, however this changed over time due to our acceptance rate, with the values settling at $0.85$ and $0.15$ respectively by trial 700, the effect of which is visible on the graph as the system begins to settle. At all times we kept the probability of modifying an expert's suggested action at $0.25$. The algorithm benefited from this greatly, producing $86$ modified actions, in several cases the base-version of the action had also been trialed, giving us the reward accrued for those actions, and on average modified actions received $14\%$ more reward than their base counterparts.

We note that the RL system takes a relatively large number of episodes to reach anything near peak performance. We expect that this is due to the relative simplicity of our tools – there is a vast range of literature on improving the performance of RL algorithms in various ways, providing a great deal of potential insights that could be implemented to improve the algorithm we use here, and these are all avenues ripe for investigation, but are left to future work. In the RL work of Stone [83] in RoboCup Soccer, the Q-Learning and SARSA algorithms (along with many other insights and engineering work) are run for upwards of 40 hours, whereas our experiments were run for around 20. At the 20-hour mark our performance is almost identical to that of Stone

### 7.4.2 Starcraft

For the Starcraft experiment we were forced to design our own reward function, as we had no pre-existing function to borrow as there was in RoboCup. That said, the Vulture Kiting task can be seen as having a very similar reward function to the one we used in RoboCup – the agent must prolong the episode length, though while dispatching enemies. In the case of this task, we include a reward for dispatching an enemy as well as a reward for behaviour that prolongs the episode. The agent must ultimately learn to balance the need to stay alive with the need to accrue points, and as the number of enemies rises both of these tasks become increasingly more difficult. Again we compare the same three systems as before,

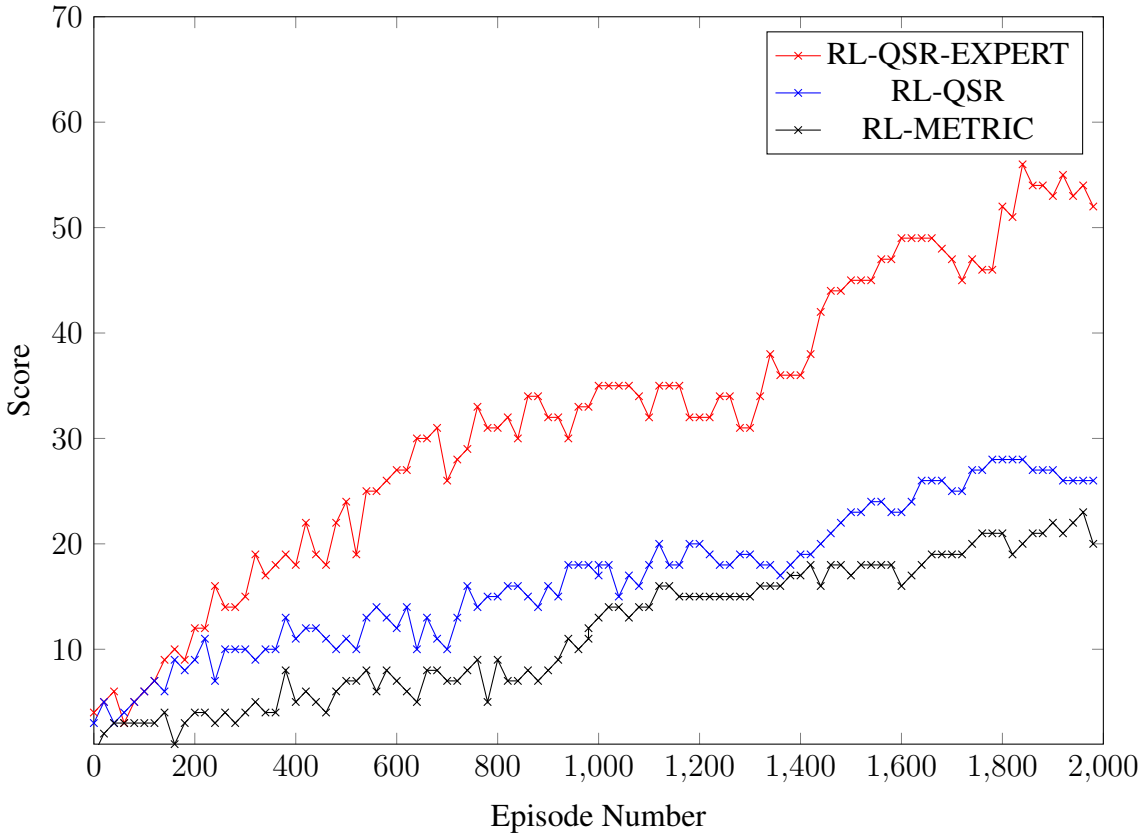


Figure 7.2: Results of Reinforcement Learning experiments in the Starcraft Vulture Kiting domain.

In this work we used hand-coded reward functions, but *learning* a reward function would also be possible. The field of Inverse Reinforcement Learning [59] seeks to learn reward functions where none is explicitly given. The work of [1] attempts to do this, where a reward function is learned by observing an expert agent in various domains – the Grid World domain and a car driving simulation. Interestingly the policy learned is guaranteed to be as good as or better than the actual reward function used by the expert. There exist a wide range of approaches to IRL [65] applied to an equally wide range of domains, though no applications at the time of our work to our specific domains.

# Summary, Discussion and Conclusion

## 8.1 Summary

We presented an approach to the problem of *Learning By Observation* in spatially situated domains, with our work specifically considering how environmental observations are delivered to a learner agent. We compared both quantitative and qualitative representations, using various forms of Qualitative Spatial Relations to improve the generalisability of learned models and minimise knowledge engineering. Our system is capable of reconfiguring its representation of the world in order to discover configurations of spatio-relational features that reveal the information used to guide an expert's underlying decision process. From here we can gain a more accurate state-action mapping, and predict the expert's behaviour. We showed that using this approach, it is possible to produce behaviour models of both synthetic, pre-programmed agents, as well as human-controlled agents that perform similarly to the original controller on various tasks. This allowed us to close the loop from observation to practical implementation in a real-world system. From there, we also showed how these learned models of agent behaviour can be used to jump-start a Reinforcement Learning algorithm with the knowledge of a domain expert, leading to agents that took advantage of the knowledge learned from the original expert, but which learned over time to *surpass* its performance.

Our work has made the following contributions.

- A full, integrated system for Learning by Observation using Qualitative Spatial Representations, and deploying learned behaviour models as controllers for virtual agents.
- Experimental comparisons between metric and qualitative representations across different

task configurations, using a variety of learning mechanisms. Analysis of the characteristics of QSR-transformed data, the problems it poses for learning systems, and practical solutions to those problems.

- An extension of the experimental work to include a study of learning from *Human* controllers, as well as synthetic controllers, in the application domains.

- Studies of situated agents making use of behaviour models learned from observing both synthetic and human experts. Study of the relationship between classifier accuracy and practical, in-situ performance of agents.

- An application of our system used in conjunction with a Reinforcement Learning approach, where the behaviour of expert agents is used to provide a head-start for the RL process.

## 8.2 Discussion

The system we developed is able to determine the spatial features that affect the underlying control algorithms of observed agents. In the Starcraft Vulture Kiting domain the synthetic controller, and the human controllers, base the decision to attack an enemy on the distance between the controlled unit and the nearest enemy – along with other factors such as the relative safety of performing the action – a conditional roughly of the sort $if distance < threshold : doattack$. Our system is capable of discovering this rule, and what values of $threshold$ are commonly used by the agent in frames where it executes an attack action. In the RoboCup Soccer domain, determining when to kick the ball is a similar task. These are the *core* actions of these domains, in that they are the actions that alter the world and further the goals of the agent toward a particular task. One characteristic of these actions is that they occur relatively rarely compared to a separate set of actions we would denote as *secondary* actions, comprising such things as movement actions, such as turning. These are far more common, in our RoboCup data we found that *dash* and *turn* actions, the game's two movement actions, accounted for 76%

of total actions recorded – similarly in the Starcraft domain, in the Vulture Kiting task – where the only two actions are to move or attack – movement actions accounted for 76% of actions taken. These kinds of actions are more difficult to predict, because they typically have a high degree of variation, and and are executed in a wide variety of states. Alternatively, the *core* actions occur in states that are more easily identifiable – for instance in RoboCup the agent will not kick the ball unless the ball is directly in front of it. The *kick*, *tackle* and *catch* actions are the actions most systems predict with the highest degree of accuracy because of this effect. There are various formulations of the action that the agent might take – passing to a team-mate, shooting at a goal etc. – and these are similarly differentiated by the same process. This underlying structure in the task is evident in all of our learning systems in some way, even the metric systems that generally perform badly, but it is the system that employs our QSR-based approach that is best able to accentuate the structure and take advantage of it, and this system achieves the highest rates of predictive accuracy on these actions. The classification performance graphs for each of the full-game teams are shown in the Appendix, with each taking up a full page, and this structure can be seen in each graph.

Further analysis of the data highlights this effect. For instance, in RoboCup Keepaway, in the final post-processed data from observing the synthetic experts, each *kick* action is used in on average 32 different states, each *dash* action is used in on average 17 different states, and each *turn* action in on average 13 different states. This highlights the difficulty of discovering state representations that cover the entire range of states in which the agent will use these actions – ie. a single representation configuration that covers all states in which the agent uses *dash* does not exist.

In our results, we observed that the C4.5 algorithm proved to be extremely fast in terms of training time, and well-suited to the implementation of the QSR representations we use, which takes the form of a boolean vector. In future work we expect that this speed could be harnessed further to build agents that learn by observation on-line in real-time, such as a RoboCup Keepaway agent learning to play in-situ as it observes its team-mates. We expect this could be accomplished initially by a simple re-training approach, rather than through modifying

any learned models on-the-fly. One problem to overcome would be the main bottleneck of our system, which is the representation selection step. This is largely composed of the repeated application of very fast operators that scale well with better computing hardware, and could easily be distributed on a cluster. As for the search itself, there no doubt exist more insights that could be incorporated in to the search so as to speed it up as well, and again could be distributed on a cluster, as Monte-Carlo methods such as the one we employ are easily distributable in such a way. Overall on-line LbO using the methods we have described would be possible, at least in the RoboCup Keepaway Domain, with some further technical work. One advantage our system has is that the underlying representation used by the learning algorithms is never destructively altered. Other learning mechanisms that benefit from some pre-processing step – feature selection, clustering etc. – may result in a permanent deformation of the state-space, as once it is transformed the original form is discarded. This can cause problems later on if, for instance, a feature thought previously to be irrelevant suddenly becomes relevant (perhaps in an edge case) – this then requires tackling the problem of how (or if, it may in fact never be detected by a system) this change is detected, and how any pre-processing steps should be reapplied to take the new worth of the feature into account. Using a re-training approach this is in principle straightforward for most learners, but the difference in training times may make this a more complicated choice.

The work of [81] discusses the problem of ad-hoc teamwork, whereby an agent must co-operate with teammates it has never seen before, cannot communicate with, and who all might follow different policies, potentially in a task it has no experience of itself. Here we expect LbO systems like ours to be key to allowing an agent to meaningfully contribute to a team effort by mimicking the behaviour of its teammates. In our own evaluation, we stick to mimicking a single set of behaviours in limited domains, however there is no reason why, in more complex scenarios, a LbO system would not be able to fulfil a multitude of team roles, perhaps even utilising a behaviour model that is the union of many different observed behaviours. In our work, we found that when we separated out the human behaviour models, and learned from them individually, rather than collecting them all up in to one single training trace, the performance

of the system was slightly higher. But this is not always the case for every LbO system. For instance, in the 2014 Computer Poker Competition [49], the winning agent used a model trained on observations of a range of previous competition winners. The system thereby surpasses the performance of individual experts. This was not the case for our system. One explanation might be that in the domain we worked in, there may be many similar states in which two agents execute different actions. Training the model with both of these data points causes ambiguity in the model, and decreases performance. In the work, we showed a way of handling training data by selecting only those traces that provide *new* information to the system. We expect the right way of combining training sets from different experts is related to this technique – instead of simply compiling the expert data into a single monolithic training set to be consumed by the learning system, the training set could instead be used to fill gaps in the agent's knowledge in an on-demand way. So, when an agent encounters a state it is unfamiliar with, it would determine whether there is an example in the data set already, and select the behaviour of a *specific, single* expert. This avoids the problem of training a classifier with multiple, conflicting examples. Another approach to doing this, which would make the distinction easier, would involve estimating the value of a reward function for a given state-action pair, so ideally the agent could make the selection based on the estimated reward for a particular action, and discard others.

### 8.2.1 Weaknesses

While our system is successful in predicting and mimicking the behaviour of several RoboCup Soccer full-game teams, it is unsuccessful in mimicking the behaviour of several others. Particularly, the predictive accuracy results in Figure 10.10 for the WrightEagle team and Figure 10.8 and Figure 10.6 for Helios and DreamWing (shown in the Appendix) are particularly poor, with our system achieving a predictive accuracy of around 39% on WrightEagle, 45% on Helios, and 43% on DreamWing. When we investigate the Technical Reports associated with these teams however, we can gain some insight as to why. DreamWing in particular is very sparsely documented, but the system does use Q-Learning to improve the skills of its players. It is unclear if this Q-Learning takes place within the confines of a specific game, and is then discarded for

the next game, is kept and built upon from game-to-game, or if the system is trained prior to being launched. In our evaluation we simply took the system as it was provided, along with all of its data files, which may have featured pre-learned policies. Regardless, our system struggles to predict the behaviour of this team. When analysing the data by hand, we determine that this is because the team seems to behave in a very unpredictable way – there are certainly set formations and other hallmarks of an intelligent system controlling the agents, however they often behave in very erratic, yet successful, patterns. DreamWing is a very high-performing team in RoboCup Tournaments, and so perhaps this unpredictability is part of it's key to success. We have similar issues predicting the behaviour of the Helios team. Further analysis shows that the system uses an on-line tree-search algorithm to search through the space of possible sequences of shots a player could make, as well as a system that oscillates between different types of tactics. Various heuristics to evaluate states are also used. One possibly intentional effect of these kinds of systems is to reduce the ability of opposing agents to predict the behaviour of the Helios player. The system is again a highly performing agent that is often found at the top of RoboCup competition leaderboards. The WrightEagle system is the champion RoboCup system, and in fact uses on-line behaviour modelling in order to guide the behaviour of its own players and to make high-level strategic decisions, and compiles various other tactical-level systems such as localisation and pass evaluation. The common theme between these systems appears to be high-level planning and strategic behaviour that is determined by systems that are either distributed between the squad, or dictated by an external agent – RoboCup teams also have a Coach agent that can issue instructions to individual players. These naturally involve the kinds of things our system is unable to detect – specifically internal state, and communication between agents. These problems lead to difficulty in predicting the next moves of players, as behaviour is then often heavily influenced by other agents that are *in communication* with the player, rather than action being influenced by what the player can observe and its own internal decision process. Further, in this way, the behaviour of an agent is influenced by state it may not even be able to observe, which is the entirety of what we base our behaviour models on, but which the communicating agent can. This communication could be intercepted and

incorporated into the information included in our behaviour models, but this is left for future work.

A particularly interesting case is the Axiom team. In the technical description of the team, the system purports to utilise an on-line Genetic Algorithm to evaluate spots on the pitch for the suitability for making shots at the goal, as well as a Neural Network for pass selection. Our system achieves only a 53% success rate in predicting the behaviour of the agents in this team, but this is still far higher than we might expect for a system that uses relatively advanced techniques to produce complex, on-line behaviour. Observation and study of the team reveals that the above techniques appear to be used sparingly, and that our system is able to replicate a degree of the supporting behaviour, which is mainly hand coded, along with some minor skills the agent exhibits.

Our Reinforcement Learning agents ultimately surpass the performance of the experts from which the predictive models that were made part of the RL algorithm were gathered. While this produces encouraging results in our laboratory experiments in a simulated environment, it is far from the kind of result that would lead us to immediately install such algorithms on expensive robot hardware or similar. The reason being that there still exists a large lead-up period before the agents are explicitly no longer engaging in relatively dangerous behaviours. Since, even while being guided by the expert models, they have the capacity to explore the state-space and engage in stochastic behaviour. Our system simply engages in the most structured form of this behaviour.

One weakness of our work with RL is that our approach is ultimately missing a complementary piece of work on learning reward functions from the kind of limited expert traces that we train our learning system with. This simultaneous learning of a reward function *while* extracting expert behaviour would provide a whole new set of insights to apply to our system in order to judge and evaluate behaviours. We expect that success in this area would come from learning reward functions from observation of a *large range* of experts, or else we may wind up with problems regarding determining whether expert behaviour is in fact beneficial, detrimental, a mistake etc. a system without this ability to judge and evaluate behaviours can only blindly

mimic what it has seen, and cannot hope to evaluate or improve its capabilities.

Our work was only evaluated in simulated domains, though RoboCup features many of the same problems as real-world domains such as noisy observations, partial observability and stochastic actions, it also makes several assumptions that make tasks easier. A continuation of this work should look towards applying the principles of our work in more complex domains, and especially domains where action labels are not as freely available (or available at all) as they were in the domains we worked in. Human activity domains are well-suited for this sort of work. In our view, the work we have done could be extended in three major ways:

- Extension to Human activity domains – observing Humans directly, engaging in Human activity prediction and mimicking of Human activities.
- Inclusion of concurrent methods to learn task reward functions from expert data. This itself opens up new approaches to the LbO problem.
- Learning of actions from the ground-up, with no prior information than what is observed. There exists already work on learning event classes using QSRs [79] which could be adapted.

## 8.3 Conclusion

Designing an AI system to learn from the entities it observes is a problem that poses both significant theoretical and engineering issues. The system we have worked to develop is just one data point in a vast space of possible systems that could be developed to address the LbO problem, with countless more possible domains in which to apply the work. The need for systems that can solve these problems will only increase as it becomes more and more crucial for Robots and AI systems to understand the activities and behaviours of the things they observe, such as Humans, Animals, and other Robots. Towards this, our sincere hope is that this work and the ideas we have tried to communicate will provide a solid base for future work, just as this work builds upon and applies the work of those who came before us.

> ”We can only see a short distance ahead, but we can see plenty there that needs to be done.” -Alan Turing

# Appendices

These appendices contain the results of our full-game experiments with RoboCup soccer. They are the classification results of the RoboCup experiments detailed in chapter 6.

# RoboCup Full Game Performance Graphs

## 10.1 AUA 2D

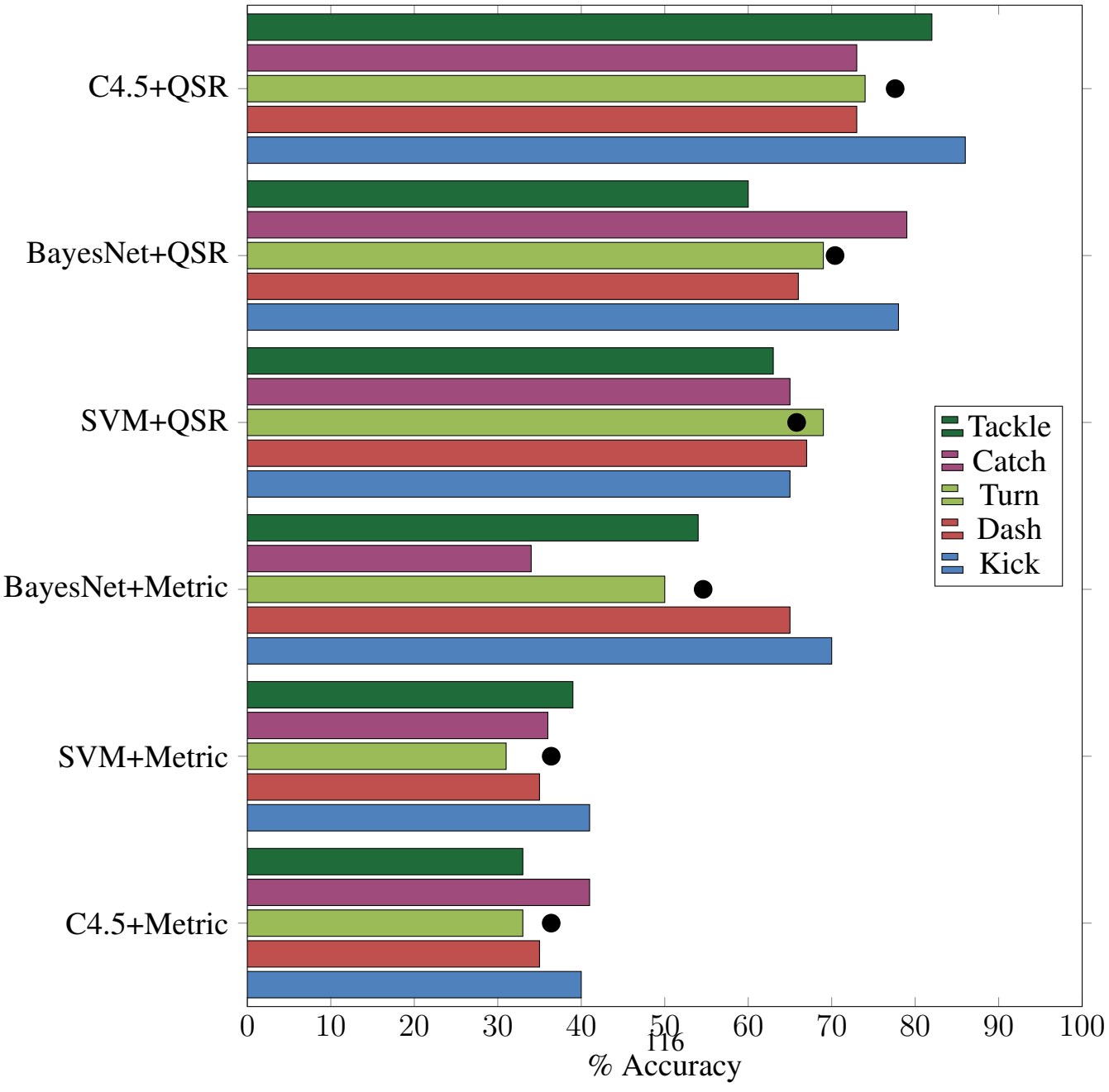


Figure 10.1: Full-game classifier comparison for the AUA 2D Team

## 10.2 AUT

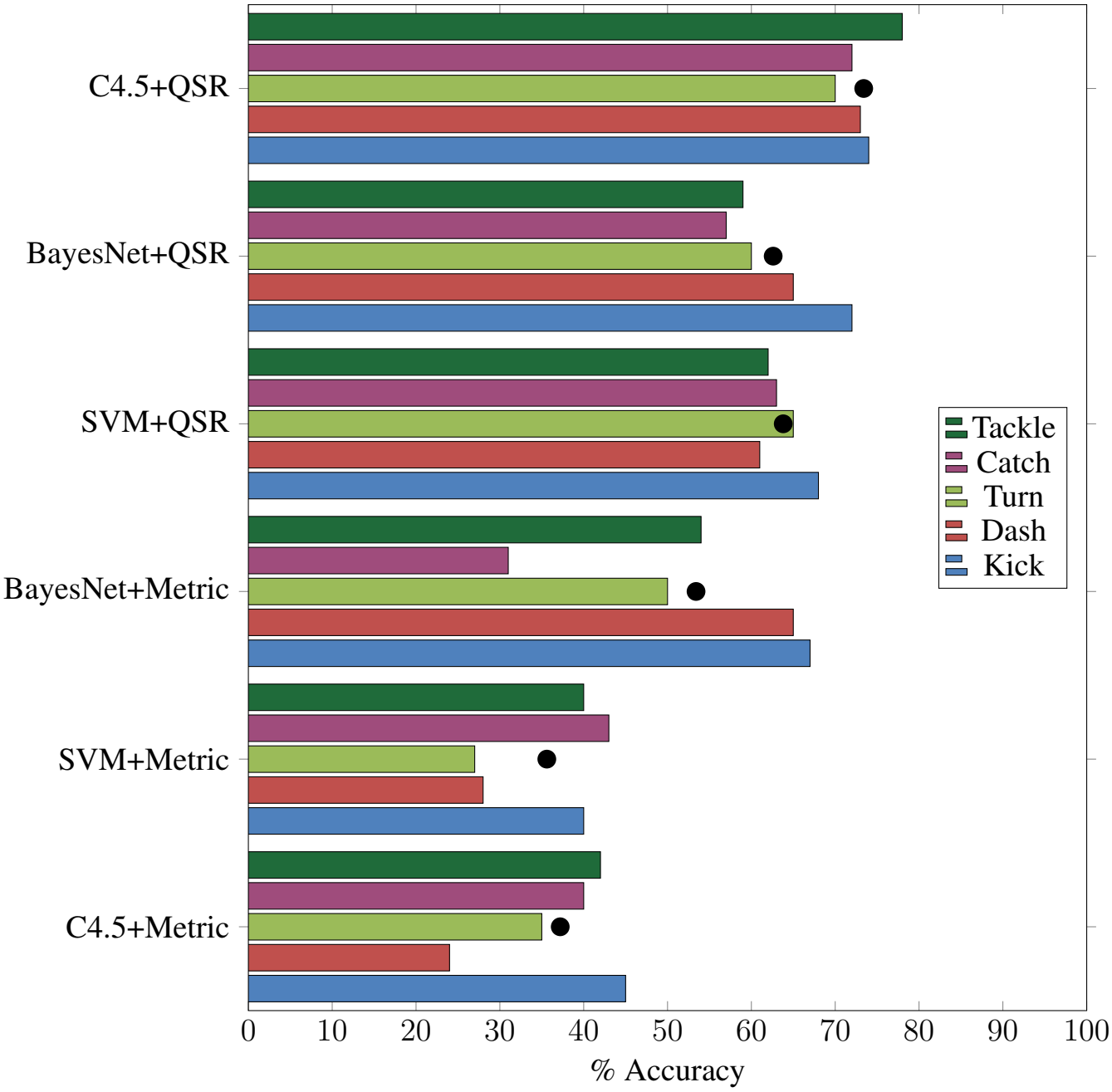


Figure 10.2: Full-game classifier comparison for the AUT Team

## 10.3 Axiom

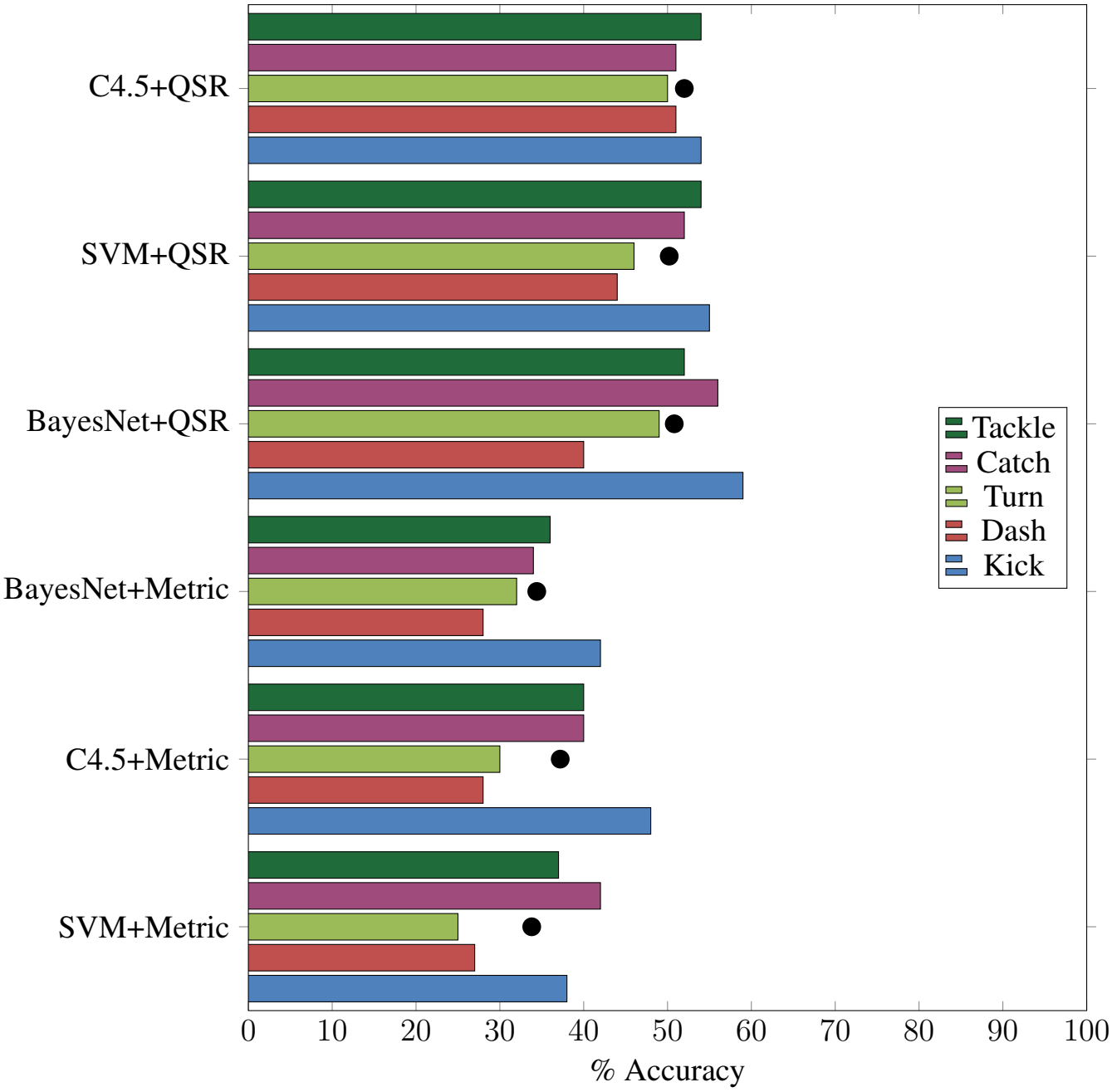


Figure 10.3: Full-game classifier comparison for the AXIOM Team

## 10.4 Borregos

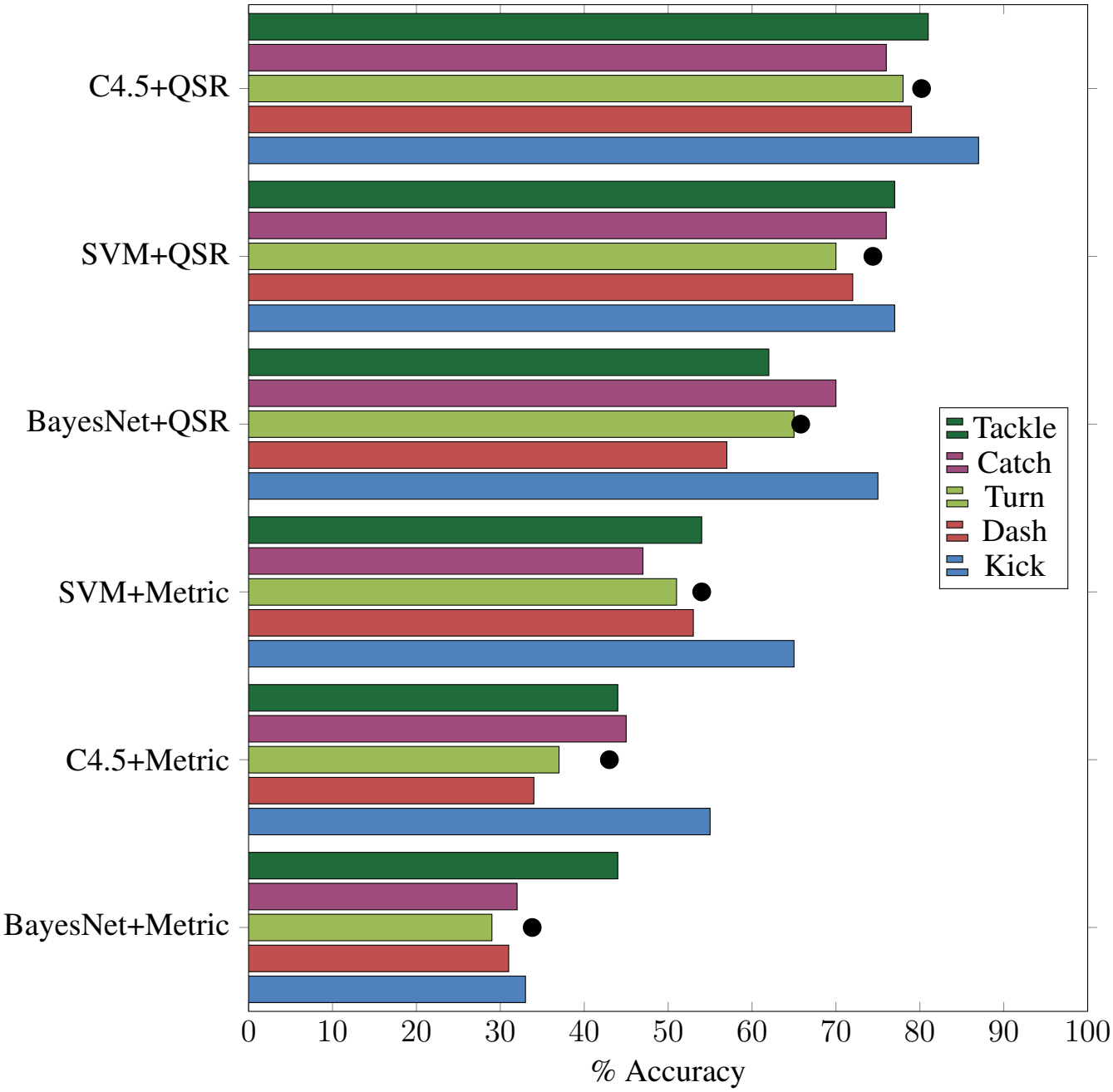


Figure 10.4: Full-game classifier comparison for the Borregos Team

## 10.5 CSU Yunlu

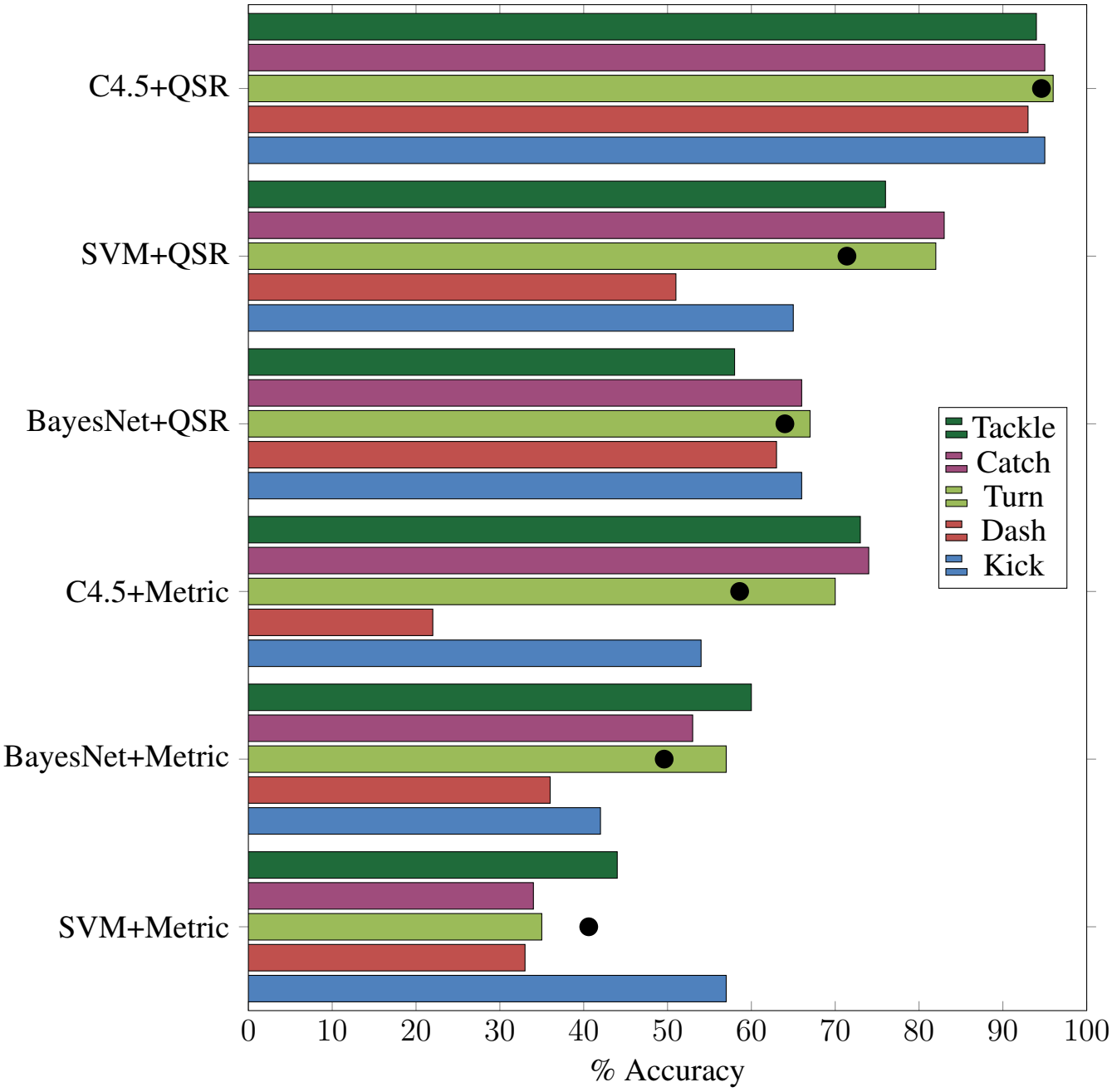


Figure 10.5: Full-game classifier comparison for the CSU Yunlu Team

## 10.6 DreamWing

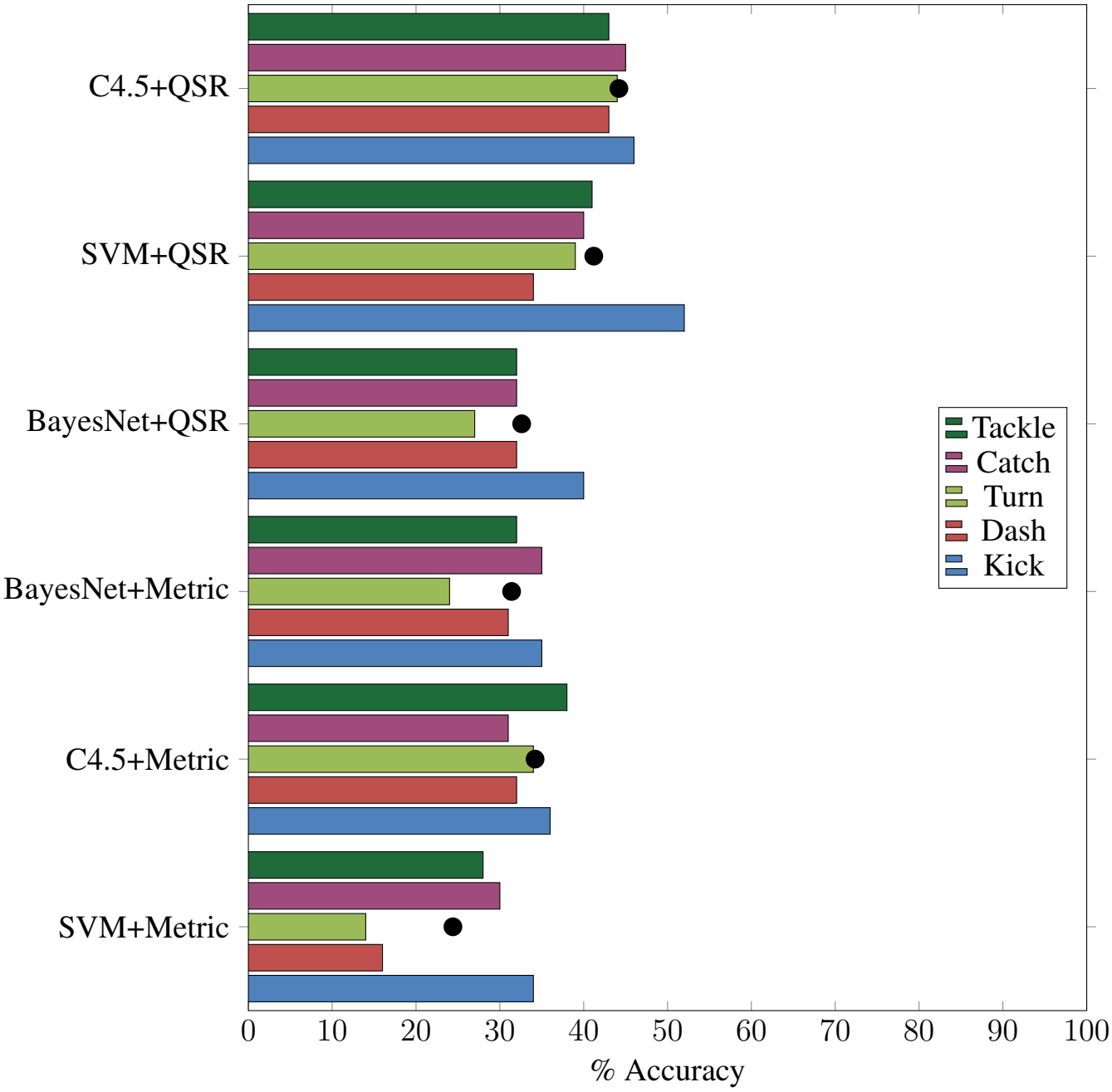


Figure 10.6: Full-game classifier comparison for the Dreanwing Team

## 10.7 GDUT

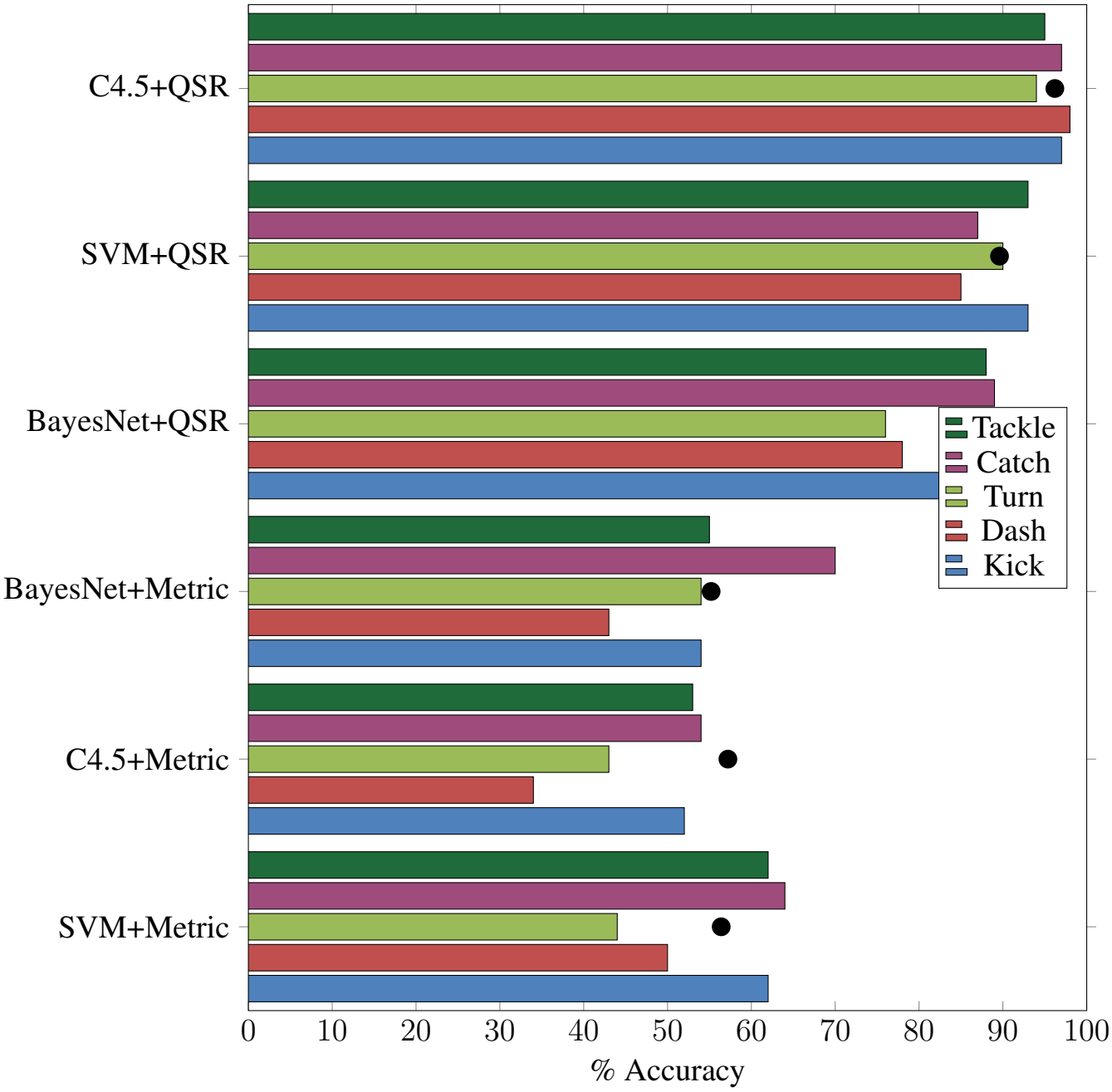


Figure 10.7: Full-game classifier comparison for the GDUT TiJi Team

## 10.8 Helios

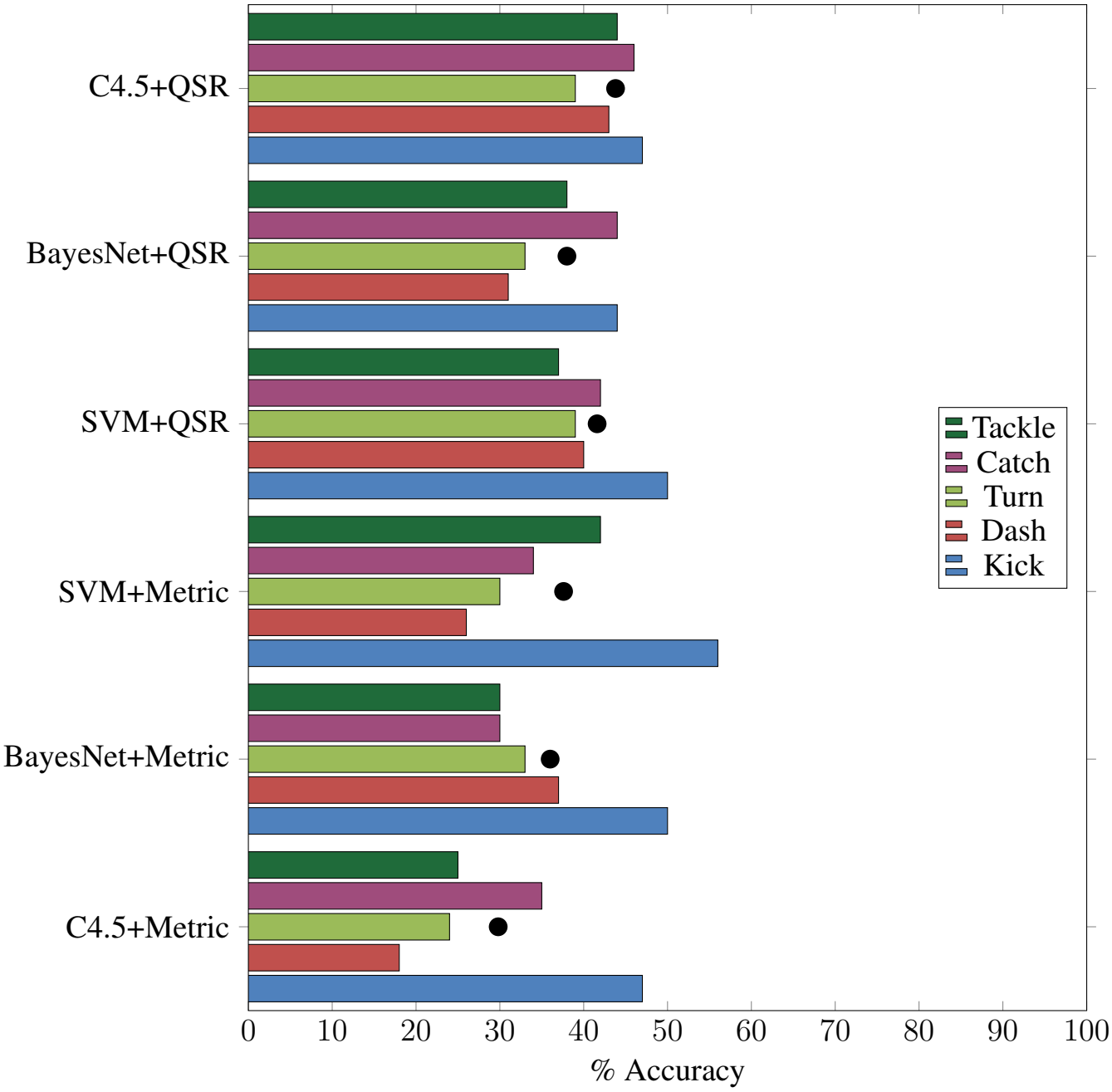


Figure 10.8: Full-game classifier comparison for the Helios Team

## 10.9 Legendary

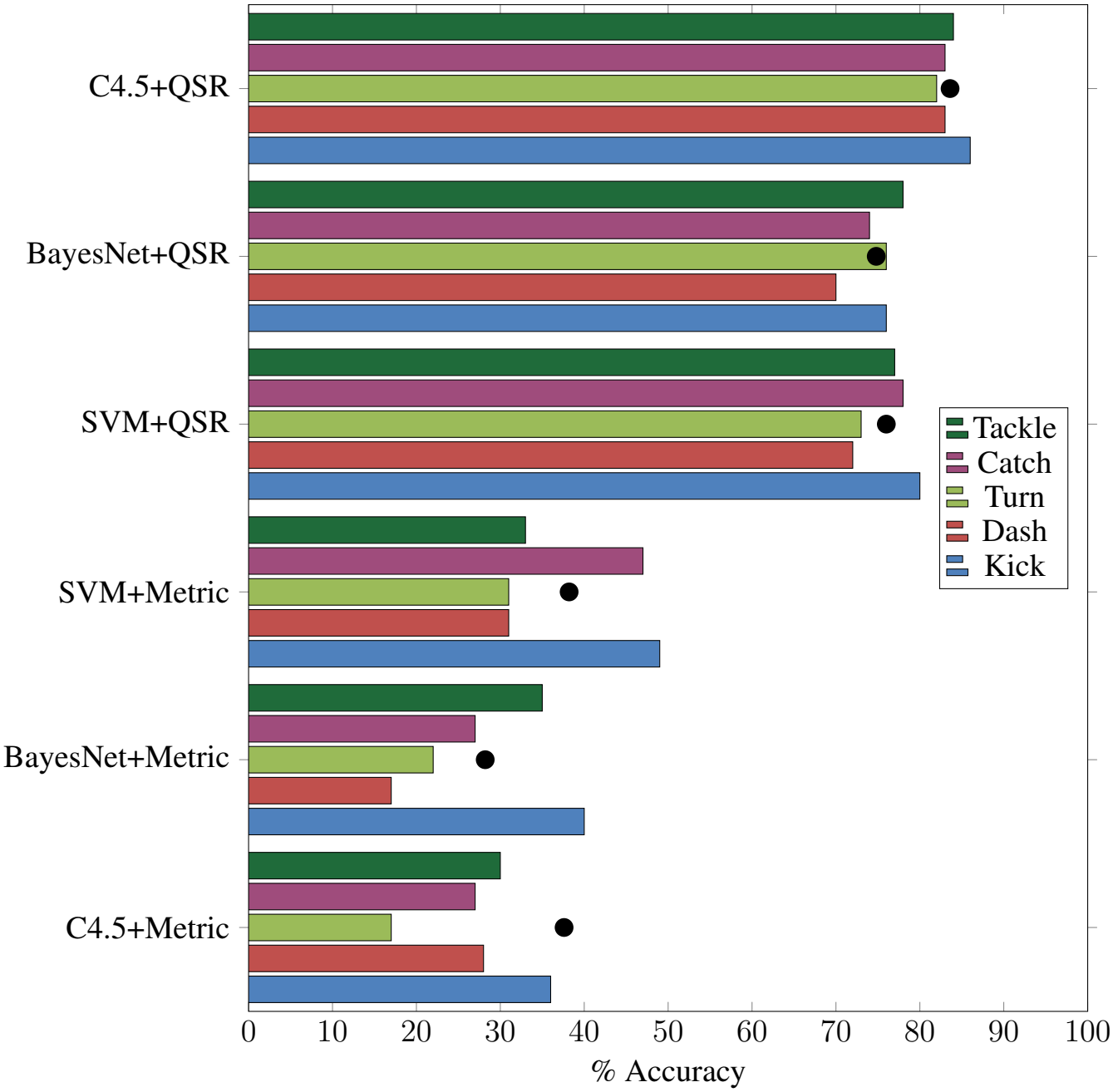


Figure 10.9: Full-game classifier comparison for the Legendary Team

## 10.10 WrightEagle

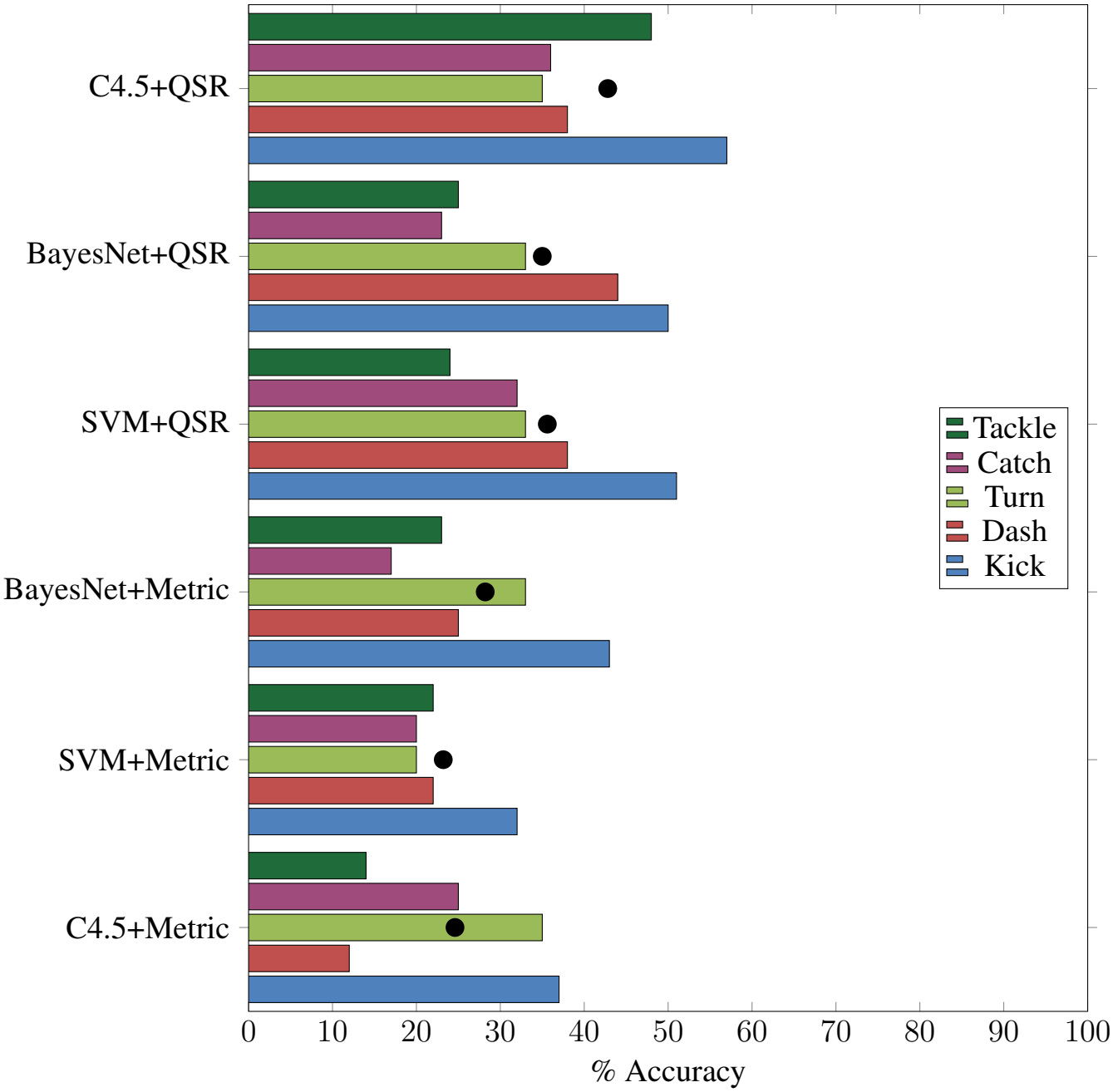


Figure 10.10: Full-game classifier comparison for the WrightEagle Team

# List of References


[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pages 1–8, 2004.

[2] Baris Akgun, Maya Cakmak, Karl Jiang, and Andrea L. Thomaz. Keyframe-based Learning from Demonstration: Method and Evaluation. *International Journal of Social Robotics*, 4(4):343–355, 2012.

[3] Hidehisa Akiyama, Tomoharu Nakashima, and Katsuhiro Yamashita. Helios2013 team description paper. *RoboCup*, 2013.

[4] Ricardo Aler, Jose M. Valls, David Camacho, and Alberto Lopez. Programming Robosoccer agents by modeling human behavior. *Expert Systems with Applications*, 36(2):1850–1859, March 2009.

[5] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[6] Jacques G Amar. The Monte Carlo Method in Science and Engineering. *Computing in science & engineering*, 8(2):9–19, 2006.

[7] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[8] Aijun Bai, Guanghui Lu, Haochong Zhang, and Xiaoping Chen. WrightEagle 2D soccer simulation team description 2011. *RoboCup Soccer Simulation 2D Competition, Istanbul, Turkey*, 2011.

[9] Paul Bakker and Yasuo Kuniyoshi. Robot see, robot do: An overview of robot imitation. In *AISB96 Workshop on Learning in Robots and Animals*, pages 3–11, 1996.

[10] Ardhendu Behera, DC Hogg, and AG Cohn. Egocentric activity monitoring and recovery. *13th Asian Conference on Computer Vision*, pages 7–9, 2013.

[11] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *Pattern Analysis and . . .* , (1993):1–30, 2013.

[12] Yoshua Bengio, Aaron Courville, and Pierre Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[13] Matthew Brand, Nuria Oliver, and Alex Pentland. Coupled hidden markov models for complex action recognition. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 994–999. IEEE, 1997.

[14] Stephen P Brooks and Byron JT Morgan. *Optimization using simulated annealing*. JSTOR, 1995.

[15] Michael Buro and David Churchill. Real-time strategy game competitions. *AI Magazine*, 33(3):106, 2012.

[16] Michael Buro and Timothy Furtak. RTS games as test-bed for real-time AI research. In *Proceedings of the 7th Joint Conference on Information Science (JCIS 2003)*, pages 481–484, 2003.

[17] Douglas R. Caldwell. Unlocking the Mysteries of the Bounding Box. *Coordinates: Online Journal of the Map and Geography Round Table, American Library Association*, (2):1–20, 2005.

[18] Eliseo Clementini, Paolino Di Felice, and Daniel Hernández. Qualitative representation of positional information. *Artificial intelligence*, 95(2):317–356, 1997.

[19] Adam Coates and Andrew Y Ng. Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*, pages 561–580. Springer, 2012.

[20] AG Cohn, Brandon Bennett, John Gooday, and NM Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 316:275–316, 1997.

[21] Anthony G Cohn and Jochen Renz. Qualitative Spatial Representation and Reasoning. *Artificial Intelligence*, 1:1–47, 2007.

[22] Randall Davis, Howard Shrobe, and Peter Szolovits. What Is a Knowledge Representation? *AI Magazine*, 14(1):17, 1993.

[23] Matthias Delafontaine, Anthony G. Cohn, and Nico Van de Weghe. Implementing a qualitative calculus to analyse moving point objects. *Expert Systems with Applications*, 38(5):5187–5196, May 2011.

[24] Ethan Dereszynski, Jesse Hostetler, and Alan Fern. Learning probabilistic behavior models in real-time strategy games. *Artificial Intelligence and Interactive Digital Entertainment*, 2011.

[25] Hans K G Fernlund, Avelino J Gonzalez, Michael Georgiopoulos, and Ronald F DeMara. Learning tactical human behavior through observation of human performance. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics*, 36(1):128–40, March 2006.

[26] Stephen Flinter and Mark T Keane. On the Automatic Generation of Case Libraries by Chunking Chess Games. *Proceedings of the 1st International Conference on Case Based Reasoning (ICCBR-95*, pages 421–430, 1995.

[27] MW Floyd. *A General-Purpose Framework for Learning by Observation*. PhD thesis, Carleton University Ottowa, 2013.

[28] MW Floyd and Babak Esfandiari. Building Learning by Observation Agents Using jLOAF. *Workshop on Case-Based Reasoning for Computer Games: 19th international conference on Case-Based Reasoning*, pages 37–41, 2011.

[29] MW Floyd, Babak Esfandiari, and Kevin Lam. A case-based reasoning approach to imitating RoboCup players. *21st International Florida Artificial Intelligence Research Society Conference*, 2008.

[30] L Frommberger. *Qualitative Spatial Abstraction in Reinforcement Learning*. PhD thesis, University of Bremen, 2010.

[31] Quentin Gemine and Firas Safadi. Imitative learning for real-time strategy games. *IEEE CIG 2012*, pages 424–429, 2012.

[32] Mohammad Ghazanfari, S Omid Shirkhorshidi, Alireza Beydaghi, Farbod Samsamipour, Hossein Rahmatizade, Mohammad Mahdavi, Mostafa Zamanipour, Payam Mohajeri, and S Mohammad H Mirhashemi. Axiom 2012 team description paper.

[33] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[34] JM Gooday and AG Cohn. Conceptual neighbourhoods in temporal and spatial reasoning. *Spatial and Temporal Reasoning, ECAI*, 94, 1994.

[35] Shane Griffith, Kaushik Subramanian, and J Scholz. Policy Shaping: Integrating Human Feedback with Reinforcement Learning. *Advances in Neural Information Processing Systems (NIPS)*, pages 1–9, 2013.

[36] Jing Guo, Haohui Huang, Jiyao Li, and Wei Chen. Gdut_tiji 2d soccer simulation team description 2012.

[37] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[38] Peter Harrington. *Machine Learning in Action*. Manning Publications Co., 2012.

[39] Mark Humphrys. Action selection methods using reinforcement learning. *From Animals to Animats*, 4:135–144, 1996.

[40] Harukazu Igarashi, Koji Nakamura, and Seiji Ishihara. Learning of soccer player agents using a policy gradient method: coordination between kicker and receiver during free kicks. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 46–52. IEEE, 2008.

[41] Noda Itsuki. Soccer server: a simulator for RoboCup. In *JSAI AI-Symposium 95: Special Session on RoboCup*. Citeseer, 1995.

[42] Mandar Joshi, Rakesh Khobragade, Saurabh Sarda, Umesh Deshpande, and Shiwali Mohan. Object-oriented representation and hierarchical reinforcement learning in infinite Mario. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 1:1076–1081, 2012.

[43] Tobias Jung and Daniel Polani. Learning RoboCup-Keepaway with Kernels. *JMLR: Workshop and Conference Proceedings*, 1:33–57, 2012.

[44] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285, 1996.

[45] Jens Kober and Jan Peters. Imitation and reinforcement learning. *IEEE Robotics & Automation Magazine*, 17(2):55–62, 2010.

[46] Savas Konur, Alexander Ferrein, and Gerhard Lakemeyer. Learning decision trees for action selection in soccer agents. *ECAI-04 Workshop on Agents in dynamic and real-time environments*, 2004.

[47] Benjamin J Kuipers, Patrick Beeson, Joseph Modayil, and Jefferson Provost. Bootstrap learning of foundational representations. *Connection Science*, 18(2):145–158, 2006.

[48] Jinsong Leng and Chee Peng Lim. Reinforcement learning of competitive and cooperative skills in soccer agents. *Applied soft computing*, 11(1):1353–1362, 2011.

[49] Michael Littman and Martin Zinkevich. The 2006 AAAI Computer Poker Competition. *ICGA Journal*, 29(3):166, 2006.

[50] Ashique Rupam Mahmood and Richard S Sutton. Representation Search through Generate and Test. In *AAAI Workshop: Learning Rich Representations from Low-Level Sensors*, 2013.

[51] Mohammadhossein Malmir, Mohammad Simchi, Shahin Boluki, and Hessamoddin Hediehloo. AUT Team Description Paper 2012. 2012.

[52] Daniel Marquez, Johny Hayworth, Braulio Chavez, Eduardo Barrera, and Alejandro Garza. Borregos robocup 2d simulation league 2012 team description paper.

[53] B W Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et biophysica acta*, 405(2):442–451, 1975.

[54] Shiwali Mohan and Je Laird. An object-oriented approach to reinforcement learning in an action game. *AIIDE*, pages 164–169, 2011.

[55] JL Montana and AJ Gonzalez. Towards a unified framework for learning from observation. *IJCAI Workshop on Agents Learning Interactively from Human Teachers*, 2011.

[56] Vlad I Morariu and Larry S Davis. Multi-agent event recognition in structured scenarios. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3289–3296. IEEE, 2011.

[57] L.G. Moseley. *Introduction to Machine Learning*. Number 4. Pitman Publishing, London, 1988.

[58] Jorge Muñoz, German Gutierrez, and Araceli Sanchis. Controller for TORCS created by imitation. *CIG2009 - 2009 IEEE Symposium on Computational Intelligence and Games*, pages 271–278, 2009.

[59] Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. *Proceedings of the Seventeenth International Conference on Machine Learning*, 0:663–670, 2000.

[60] Ricardo Parra and Leonardo Garrido. Bayesian networks for micromanagement decision imitation in the RTS game Starcraft. In *Advances in Computational Intelligence*, pages 433–443. Springer, 2012.

[61] Kris Pigna. StarCraft Cheating Scandal Rocks South Korea. *1UP News – http://www.1up.com/news/starcraft-cheating-scandal-rocks-south – accessed 1st May 2016*, April 2010.

[62] Bob Price and Craig Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

[63] J R Quinlan. *C4.5: Programs for Machine Learning*. Elsevier, 1993.

[64] L R Rabiner and B H Juang. An introduction to hidden Markov models. *ASSP Magazine,*, 3:4–16, June 1986.

[65] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *IJCAI International Joint Conference on Artificial Intelligence*, pages 2586–2591, 2007.

[66] David A Randell, Zhan Cui, and Anthony G Cohn. A spatial logic based on regions and connection. *Proceedings of the 3rd International conference on Knowledge Representation and Reasoning*, 92:165–176, 1992.

[67] Jochen Renz and Debasis Mitra. Qualitative Direction Calculi with Arbitrary Granularity. *PRICAI 2004: Trends in Artificial Intelligence*, pages 65–74, 2004.

[68] Glen Robertson. Applying Learning by Observation and Case-Based Reasoning to Improve Commercial RTS Game AI. *Eighth Artificial Intelligence and Interactive Digital Entertainment*, 2012.

[69] Glen Robertson and Ian D Watson. A Review of Real-Time Strategy Game AI. *AI Magazine*, 35(4):75–104, 2014.

[70] Houcine Romdhane and Luc Lamontagne. Reinforcement of Local Pattern Cases for Playing Tetris. In *FLAIRS Conference*, pages 263–268, 2008.

[71] Gavin A Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. Technical report, Cambridge University Engineering Department, 1994.

[72] Pourya Saljoughi, Reza Ma'anijou, Ehsan Fouladi, Narges Majidi, Saber Yaghoobi, Houman Fallah, and Saeideh Zahedi. Legendary 2012 soccer 2d simulation team description paper.

[73] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[74] Joe Saunders, Miscellaneous Imitation, Programming Demonstration, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, (September 2015):118–125, 2006.

[75] D Sculley et al. Results from a semi-supervised feature learning competition. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[76] Aa Sherstov and Peter Stone. On Continuous-Action Q-Learning via Tile Coding Function Approximation. Technical report, Department of Computer Sciences, The University of Texas at Austin, 2004.

[77] Alexander A Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In *Abstraction, Reformulation and Approximation*, pages 194–205. Springer, 2005.

[78] Kouki Shimada, Yasutake Takahashi, and Minoru Asada. Efficient Behavior Learning by Utilizing Estimated State Value of Self and Teammates. *Robot Soccer World Cup XIII*, pages 1–11, 2010.

[79] Muralikrishna Sridhar, Anthony G Cohn, and David C Hogg. Unsupervised learning of event classes from video. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1631–1638. AAAI Press, 2010.

[80] Frieder Stolzenburg. Localization, exploration, and navigation based on qualitative angle information. *Spatial Cognition & Computation*, 10(1):28–52, 2010.

[81] Peter Stone, Gal A Kaminka, Sarit Kraus, Jeffrey S Rosenschein, et al. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *AAAI*, 2010.

[82] Peter Stone, Gregory Kuhlmann, Matthew E Taylor, and Yaxin Liu. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup 2005: Robot Soccer World Cup IX*, pages 93–105. Springer, 2005.

[83] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement Learning for RoboCup-Soccer Keepaway. *Adaptive Behavior*, pages 1–50, 2005.

[84] Gabriel Synnaeve and Pierre Bessiere. A Bayesian model for opening prediction in RTS games with application to Starcraft. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 281–288. IEEE, 2011.

[85] Andrea Lockerd Thomaz and Cynthia Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *AAAI*, volume 6, pages 1000–1005, 2006.

[86] Sebastian Thrun, Anton Schwartz, et al. Finding structure in reinforcement learning. *Advances in neural information processing systems*, pages 385–392, 1995.

[87] Nico Van de Weghe, Bart Kuijpers, Peter Bogaert, and Philippe De Maeyer. A Qualitative Trajectory Calculus and the Composition of Its Relations. *GeoSpatial Semantics*, 281:60–76, 2005.

[88] Davi C De L Vieira, Paulo J L Adeodato, and Paulo M Gon. Improving Reinforcement Learning Algorithms by the Use of Data Mining Techniques for Feature and Action Selection. *IEEE International Conference on Systems Man and Cybernetics*, pages 1863–1870, 2010.

[89] Pablo J. Villacorta and David a. Pelta. Theoretical analysis of expected payoff in an adversarial domain. *Information Sciences*, 186(1):93–104, March 2012.

[90] Huizhao Wang, Weishan Huang, Li Liu, Jun Peng, and Fu Jiang. Csu_yunlu soccer simulation team description paper.

[91] Martha White. Integrating Representation Learning and Temporal Difference Learning: A Matrix Factorization Approach. *AAAI Workshop on Sequential Decision-Making with Big Data,*, (2), 2014.

[92] S. Whiteson, M. E. Taylor, and P. Stone. Empirical Studies in Action Selection with Reinforcement Learning. *Adaptive Behavior*, 15(1):33–50, 2007.

[93] Shimon Whiteson, Matthew E Taylor, Peter Stone, et al. *Adaptive tile coding for value function approximation*. Computer Science Department, University of Texas at Austin, 2007.

[94] Auke J Wiggers. Recognizing Attack Patterns : Clustering of Optical Flow Vectors in RoboCup Soccer. *Thesis. University of Amsterdam*, 2012.

[95] Jay Young. Evolutionary Learning of Goal Priorities in a Real-Time Strategy Game . *The*

*Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.

[96] Jay Young and Nick Hawes. Learning Micro-Management Skills in RTS Games by Imitating Experts. In *In Proceedings of the 10th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2014.

[97] Jay Young, Fran Smith, Christopher Atkinson, Ken Poyner, and Tom Chothia. SCAIL: An integrated Starcraft AI system. *IEEE Computational Intelligence and Games*, 2012.

[98] Huilong Zhang, Fajun Zhao, Yanan Fan, Changning Huang, Yu Gan, and Binbin Ruan. DreamWing2D Simulation 2D Team Description Paper for RoboCup 2012.

[99] Junhai Zhang, Yanan Li, Wangfei Zhang, and Runmei Zhang. Aua2d soccer simulation team description paper for robocup 2012.

[100] Richard Zhao and Duane Szafron. Learning Character Behaviors using Agent Modeling in Games. *AIIDE*, pages 179–185, 2009.