Multimodal Behaviour Learning for Mobile Robots

Evgeni Sergeev 10428622

Supervisor: Assoc. Prof. Dr. Thomas Bräunl

Centre for Intelligent Information Processing Systems

School of Electrical, Electronic and Computer Engineering

THE UNIVERSITY OF WESTERN AUSTRALIA

Achieving International Excellence

October 31, 2008

Letter to the Dean

1 Gerda Court
GREENWOOD WA 6024

31 October 2008

The Dean
Faculty of Engineering, Computing and Mathematics
The University of Western Australia
35 Stirling Highway
CRAWLEY WA 6009

Dear Madam

I submit to you this dissertation entitled “Multimodal Behaviour Learning for Mobile Robots” in partial fulfilment of the requirement of the award of Bachelor of Engineering.

Yours faithfully

Evgeni Sergeev

Abstract

We present a brief analysis of robotic programming techniques in the context of service robots, deriving the requirements of such systems to incorporate a high-level planner, a behaviour-based layer responsible for addressing unstructured problems, and machine learning modules for working with specific, complex motions which may involve many degrees of freedom. As the most important requirement for large-scale systems, we see the traceability of observed aspects of behaviour back to the modules causing these aspects.

We present an initial prototype developing these ideas in application to the task of navigation in a cluttered, maze-like environment. The prototype uses an expandable set of event monitors, the states of which regulate which behaviour rules are enabled at any given time. Behaviour rules have both a level and a weight: levels for unconditional precedence over other rules, when such is required, and weights to facilitate an arbitrary selection of behaviours which are at the same level. The system allows behaviour rules to be added and modified rapidly. Record-and-replay functionality is provided, for reviewing the actions of the robot and the reasons for taking particular actions—these are easily and explicitly observable through the user interface.

A Q-Learning module is applied to the task of basic trajectory learning. It is shown that given high-quality sample output by a human operator, this unsupervised learning module learns faster than otherwise.

Contents overview. There are three chapters:

1. *Background* explains the motivation for the study.

2. *Multimodal behaviour learning* discusses the logic of the approach and the details.

3. *Experimental evaluation* describes the experiments and the results.

Acknowledgements

I would like to thank firstly my supervisor, Dr. Thomas Bräunl, for setting a project on this interesting and important topic and for providing guidance during our weekly meetings. Also thanks to Dale Morey and to Lim for sharing ideas on behaviour-based robotics and programming on the EyeBot platform.

I am also very much indebted to the authors of the EyeSim simulator, which has been an invaluable basis supporting my project.

Contents

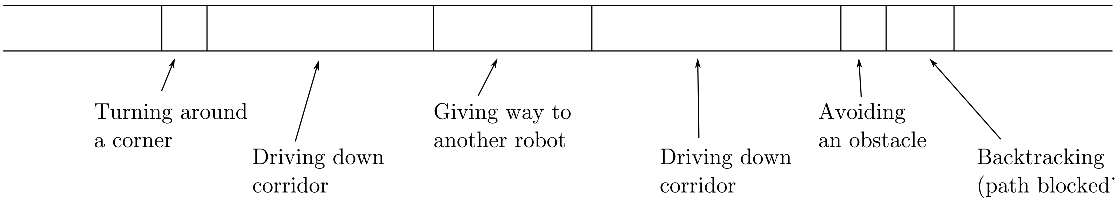
Introduction

The overarching goal of robotics is to create mechatronic systems which assist humans and are cost-effective overall.

If we plot a timeline for a hypothetical service robot working on a hypothetical generic task, we would observe phases such as the following:

...	Driving to object	Fine positioning	Fastening harness	Navigating with object in tow	Positioning relative to storage bay entrance	...

These are stages of an unambiguous algorithm, where each stage fits between two neighbouring stages in a definite order. However, if we consider just one of these, for example "Navigating with object in tow", in detail, then we are likely to observe patterns as follows:



These occur in response to occurrences in the environment, which have been anticipated, but the time of occurrence of which is unpredictable.

Furthermore, if we examine one of these patterns of motion, or even the "Fastening harness" or "Fine positioning" stages of the algorithm, we see that they are single-purpose motions of considerable complexity, in which the trajectory of the robot (or its manipulators) is affected in real time by the environmental aspect it is controlling. (Fine positioning is **relative** to the object; its relative position must be continually sensed by the robot as it adjusts its own.)

Much research is concerned with designing effective frameworks which would allow robot programmers to group and sequence many simple actions, to allow them to express how the robot is ought to behave, and to be able to do so as promptly as they would explain the same to a human being.

The transfer of "skills" to robots has largely been addressed by either placing tight constraints on production environments, or the use of simple behaviour-based robotics in less-structured environments.

However, the creation of a highly versatile and scaleable behaviour management system has proved to be a major challenge. Attempting to scale existing systems to moderately complicated tasks is problematic. The simple vision of being able to train a general-purpose robot to perform a new unstructured, repetitive task in a matter of hours is in sharp contrast with reality: months are necessary rather than hours.[1]

[1] The ASIMO robot has been in development for over twenty years and is estimated to cost close to $1 million per unit, and while it is able to run and even dance, its ability to manipulate objects is still very limited [25].

Chapter 1

Background

1.1 Summary of the problems

We shall now briefly outline the problems preventing robotics paradigms from realising the vision.

Software for highly structured environments is actuator-centric. Sensors are used as ON/OFF switches, detecting specific and highly anticipated events (e.g. a beam indicating the presence of an object on a conveyor-belt). In unstructured environments, the programming style must change to be sensor-centric.

Deliberative planning systems for unstructured environments perform a substantial amount of sensing, which they use to build a world model. This approach is criticised for the empiric difficulty of building an accurate-enough world model, and for the unresponsiveness of such systems

In **behaviour-based systems**, many independent functional units may be said to maintain their own, partial world models of only the aspects of the environment relevant to their function. Behaviour-based systems are successful in simple-to-moderate applications in unstructured environments. Scaling them to more complicated applications is problematic precisely because the behaviours are so independent. Pure behaviour-based systems often seem to lack a governing sense of purpose, (the kind of situational awareness that is afforded by maintaining a world model), while responding to stimuli elegantly.

Hybrid systems combine behaviour-based (also called *reactive*) modules with deliberative planners. These systems have been very successful. Once again, however, the best of them have not reached a level of effectiveness that advanced service robots would require.[1]

What are the weaknesses in these systems?

An illuminating example might be to consider a representative service robot task, such as the use of a human-like hand end-effector, in combination with video cameras, to manipulate textile (in predetermined ways). The use of pressure feedback and visual feedback are essential. The end-effector could have of the order of 20 degrees of freedom. It seems that the necessary motions cannot be represented as a text-based program elegantly.

Machine learning endeavours may be seen as a pursuit of systems which program themselves, either through trial-and-error (in the case of unsupervised learning), or of systems which can be programmed via non-text-based methods (supervised learning). The machines are informed of desirable outputs, and are expected to produce them. Where robots are concerned, these outputs are motions, which depend non-trivially on the system inputs.

Machine learning approaches can be very successful for narrow applications (e.g. movement with one, unchanging, goal), even when the required output is complex (involving many degrees of freedom).

We define our problem as follows:

[1] In over a decade, no advanced service robots have emerged onto the market, indicating that the technology is still insufficient.

Deliberative planners, behaviour-based robotics, machine learning are each good for solving different problems. The timely synthesis of a control system for a service robot would require the use of all these techniques and it would require for them to be well organised in a systematic, scaleable way, **such that any discrepancy in its behaviour against the expected behaviour could be unambiguously traced to a specific component**.

It seems that the key to the problem is that we are dealing with **large** systems. In an analogy to software systems, we are now able to work with software programs of enormous size, debugging and extending them with speed which is almost independent of the size of the rest of the codebase, as a result of the introduction of object-oriented programming techniques, which essentially did nothing more than to compartmentalise previously available functionality for manageability.

The key enabler seems to us to be the ability (if OO techniques are used properly) of the programmer to trace the source of a particular program action to the corresponding place in the code in a reasonable amount of time. Hence we emphasise in our problem statement above, what seems to be the parallel in robotics. Whereas software is concerned with changing variables and performing calculations, robotics deals with sensing and motion, and these, we believe, can also be organised scaleably.

1.2 Literature review

1.2.1 Service robots

To help direct our study into an area of relevance, we have studied two major roadmaps of robotics: the EURON (European Robotics Network) Roadmap [13] and the US Department of Energy Robotics and Intelligent Machines Roadmap [14]. The highlights identified include the drive towards modularisation and standardisation of interfaces for robotics hardware, but more directly important to us:

- The expansion of the robotics industry into **service robot** applications, where unstructured environments are unavoidable;
- The state of the art in robotic **vision** being identified as a key bottleneck;
- The emerging necessity to develop methods to control generic **multi-purpose end-effectors** (especially human-like hands, as so many aspects of the human environment are designed for them); and
- Applications such as the manipulation of soft materials necessitating a significant volume of high-quality sensory feedback.

Both multi-purpose end-effectors and the manipulation of soft materials require the ability to handle many degrees of freedom, coordinated in such a way that programming in terms of individual degrees of freedom becomes impractical.

The EURON Roadmap reminds us that previously unavailable opportunities are opening up in this area of technology, due to the constantly reducing cost and the improving performance of embedded systems reaching a stage where many of the applications considered are approaching commercial feasibility.

1.2.2 Behaviour-based robotics

Prior to the emergence of behaviour-based robotics, common robot control systems could be described as *deliberative planners* or *symbolic manipulators* (often referred to as “traditional” systems in behaviour-based robotics literature): these systems attempted to build inside themselves a world model, plan a route within that model, and then put the plan into action. Responsiveness was poor, limiting robots to low velocities. A system of such a design would be centrally controlled by the sequential symbolic manipulator, which is decoupled from direct sensing and only reads sensor data at times determined by its algorithm. It was realised that to achieve better responsiveness, changes in sensor readings had to play a central, not a background, role in driving control. For responses to such changes to be fast, the processing associated with each important sensory change has to be limited, hence many agents would be necessary to account for many possible types of changes. An in-depth criticism of symbolic manipulators can be found in Brooks' “Elephants Don't Play Chess” [16].

Behaviour-based robotics is an approach to robotics programming, where independent functional units, called behaviours, control the robot at different times. The arbitration, or selection, of which behaviour is active at any given time, is the primary point of difference between existing behaviour-based architectures. The early Subsumption architecture allowed only fixed precedence rules: some behaviours could suppress or inhibit others [15]. Ronald Arkin describes a number of differing approaches in the book *Behaviour-Based Robotics* [2], in particular his own Autonomous Robot Architecture, in which behaviours output one vector each (specifying the best direction for the robot to move in, from that behaviour's functional perspective). The vectors are weighted, with the weights being provided by a higher-level planner, and then added together. The resulting vector is then forwarded to the motor control system. To override the effect of other behaviours, a single behaviour in this system would output a vector of large magnitude—for example, an obstacle-avoidance behaviour in close proximity to an obstacle would thus take over the system to avoid an accident.

Our view is that the relationship between behaviours should be appropriate to the application and the situation: outright suppression, cooperative vector summation and random selection are all useful types of relationships and should be allowed for, especially now, when the hardware available to us can accommodate them easily.

1.2.3 Hybrid systems

Hybrid systems aim to combine the strengths of both paradigms: the “farsightedness” of deliberative planning systems and the speed of response of reactive systems. While reactive systems are well-suited to respond to clear stimuli and simple patterns of stimuli, planning further ahead is problematic, because the number of cases that require associated responses becomes intractable rapidly. Viewing a reactive system as a type of hard-wired cache for the commonly recurring outputs of a deliberative system, and noting that there are memory (or component count) constraints on the size of such a cache, a conclusion is made that a high-level plan *must be synthesised* online [17]. In light of Brooks' argument for the use of the "world as its own best model" [16], we can liken a deliberative algorithm to the imagination of a human being, which is able to test the probable consequences of a large number of possible actions much faster than the time required to actually perform them.[2] It seems then that deliberative systems cannot be totally replaced by behaviour-based robotics. The converse is also true, as the requirement of quick reactions necessarily implies high parallelism and mutual independence of agents—which is a general definition of the parallel-reactive approach.

A representative example of a hybrid system is the 3T architecture [17]. The system consists of three tiers. Apart from a reactive layer and a deliberative planner, there is middle layer, called the sequencer, which selects sets of skills to accomplish the tasks at hand. Each type of task is associated with a Reactive Action Package—these are hierarchical modules containing the skills relevant to a particular task. *Event monitors* are used for sequencing skill sets. In T3, the planner is able to *counterplan*: anticipate the actions of the agents in the reactive layer, referred to as uncontrolled agents (i.e. uncontrolled by the planner).

Other hybrid systems of note include SOMASS [11] (where it is noted that the use of behaviours simplies the planner, while the planner is effective at ensuring the reliable execution of plans) and SSS [12].

Criticism of hybrid systems usually concerns the interface between the planner and the reactive system. The two systems are very different in nature—the planner is mostly sequential, performing the stages of a long-running algorithm in order, while the reactive system consists of many independent, lightweight units which operate

[2] Humans do play chess.

in parallel. In particular, there is the *model-coherency problem* [6], arising from conventional deliberative planners (within hybrid systems) having to **explicitly** query sensors to update their internal world models; the programmer has to ensure that such queries are timely and and at all times to be conscious that the world model might be outdated.

1.2.4 Generalisations of behaviour-based systems

The Subsumption architecture, and the family of related behaviour-based architectures outlined in [2], all share several essential properties, the properties that make them necessarily reactive. Generalisations, which we refer to as *parallel-reactive* systems, attempt to retain these properties, while providing tools similar to those used in development of deliberative planners, as these tools often allow the intentions of programmers to be expressed naturally.

One generalisation, exemplified in a system called Cerebus, is centred around the property of *circuit semantics*: a strictly feed-forward system composed of logic gates and finite-state machines guarantees that changes in premises (i.e. immediate outputs of sensory systems) propagate through the network in $O(1)$ time ("flow" through it), if the mechanism for such propagation is parallel.[3] The set of the finite-state machines is the short-term memory of the system. Higher-order semantics are derived through the use of the same finite-state machines via *role passing*. Role passing is a generalisation of deictic representation—where information about the current situation is represented by the true/false values of a great number of predicates which have been anticipated at design time. [5]

The language GRL has been designed such that valid programs in it compile to such finite-state machine networks as described above. GRL is based on a restricted subset of LISP. It allows the expression of very general parallel-reactive systems [6]. Although we do not use GRL in our work, the concepts of circuit semantics and

[3] When the parallelism is emulated by a sequential system (a microprocessor), it follows that the speed of propagation is limited by the instantaneous volume of sensory updates—not a concern in the case of Cerebus, which performs at "sensory frame-rates" [5].

finite-state machines, in the way they are used in Cerebus, are key ideas shaping our system.

1.2.5 Blackboard systems

The concept of the blackboard, used in generic multi-agent systems, has been applied to a behaviour-based mobile robot architecture, in order to scale up the capabilities of behaviours, whilst retaining the decomposition of the system into separate agents [7]. A blackboard is a shared large data structure, the information on which is used opportunistically by agents (the behaviours in this case). Some agents produce information which they place on the blackboard and which may be used by other agents. This model allows reflective behaviours to be created by having behaviours which should be monitored put useful information about their internal state onto the blackboard.In spirit, the motivation for this effort is similar to the parallel-reactive Cerebus, in which it is safe to use any predicate within the system, because it is assured that all predicates are current.

1.2.6 Machine learning for robotic tasks

Owing to the difficulties of specifying, in code, precise and complex motions relying on sensory information such as pressure feedback and the accurate position of parts, we have explored the field of machine learning (ML) applicable to mobile robotics.

The generic requirement of a machine learning system is, given a set of relevant inputs and a set of possible outputs, and provided with training time, to then be able to find the optimal output for any valid state of the inputs. Here “optimal” implies an objective, or subjective, metric on the quality of outputs, given the inputs. This metric, called a *fitness function*, is often much more straightforward to specify than the actions which maximise it (in other word, *what* has to be done is easier to specify that *how* to do it). Types of approaches are:

- *Supervised learning*, where an operator provides outputs **which the machine considers ideal** and attempts to approximate. The provision of correct examples is the process of specifying the fitness function.

- *Unsupervised learning*, where a fitness function is fully pre-specified and the machine uses trial-and-error to discover the relationship between the inputs and the desirable outputs.

Thus supervised learning is more applicable where the fitness function is elusive, but perfect examples are readily available, whereas unsupervised learning is more suited to applications where the fitness function is trivial, and perfect examples are hard to construct and must be discovered.

Mobile robotics appears to be mostly concerned with unsupervised learning. The goal of a robotic action is usually clear, while the best action to use is not. Supervised learning is more applicable to classification tasks, where a machine is required to assign classes to items, given a range of a priori correct examples.

It has been noted [18] that while human operators may not always be able to supply perfect examples of robotic control, they often provide acceptable examples, and these may be used to seed an unsupervised learning system, greately speeding up learning during the initial stages. In the later stages, an operator's contribution becomes less relevant, as the performance of the robotic system, on average, surpasses that of the operator.

In *reinforcement learning*, the *reward* plays a central role. The reward is another name for the fitness function. What differentiates reinforcement learning from other types of machine learning is the temporal nature of the process which is to be learnt and **delayed rewards**. Thus reinforcement learning is highly applicable to temporal motion control. Reinforcement learning is addressed in-depth in the book *Reinforcement Learning: An Introduction* by Sutton and Barto [9].

Q-Learning

Q-Learning has been identified as a simple and effective reinforcement learning algorithm for the control of mobile robots [18]. The aim of the algorithm is to find a *policy*: a map from states to actions such that the reward over long periods of time is maximised. It has been compared with a similar technique called Adaptive Heuristic Critic in [19] and found to be easier to implement, to have more intuitively defined parameters controlling the learning rate and to be *exploration insensitive*: convergence to optimal values is guaranteed independently of the exploration strategy (on the assumption that all state-action pairs are reached often enough). It is concluded that Q-Learning "seems to be the most effective model-free algorithm for learning from delayed reinforcement" ([19], page 251). This view is supported by Lin [4]. (*Model-free* refers to algorithms which learn a policy directly, in contrast with *model-based* algorithms, which learn the reward function and the state transition probabilities[4] and derive the policy from them; in fact model-based algorithms are generally described as needing less experience at the expense of greater computational resources, but we rely on the software simulator as an unlimited source of experience.)

The identified weaknesses of Q-Learning are a lack of an inherent capability to generalise over state and action spaces and the resulting potentially slow rate of convergence where the state or action spaces are large [19].

The algorithm is described and thoroughly analysed in [3] (Watkins and Dayan 1992). Q-Learning is based on a Markov Decision Processes (MDPs) model. In this model, there is a set of states S. At any time t (discrete, starting from an initial time $t = 0$), the system is in exactly one state; which one is known precisely. There is also a set of actions, A, one of which the system may choose. There is a set of unknown probabilities associated with each action $a \in A$ and the current state $s_t \in S$, of transitioning to each of the other possible states. The system attempts to maximise the *discounted cumulative reinforcement* (or *utility*)

$$\sum_{t=0}^{\infty} \gamma^t R(s_t) \tag{1.1}$$

where $R(s_t)$ is the constant reward given for being in state s_t for one time step, and γ is the *discount rate*, $0 \leq \gamma \leq 1$, controlled by the operator of the system.

[4] The probability of transitioning to a state s' from a state s after an action a has been taken, $\forall s', a, s$.

When $\gamma = 1$, each reward is equally important to the system, no matter when it occurs in the future. For $\gamma = 0$, only the immediate reward is important, and the algorithm reduces to taking the action with the maximum predicted immediate reward at any point in time.

It has been shown that the algorithm converges to an optimal policy [3]. A policy π is a map from the current state to an action. An optimal policy is a policy which maximises (1.1). Central to the algorithm is a table of Q-values $Q(s, a)$ (one real number is kept for each of the state-action pairs).

The algorithm proceeds as follows:

```
$s' \leftarrow$ initial state
loop:
        $s \leftarrow s'$
        With probability $\epsilon$ select action a maximising $Q(s,a)$.
                Otherwise select random $a$.
        Perform action $a$ (transition to a new state).
        $s' \leftarrow$ the resulting state
        $r \leftarrow$ reward
        $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma\ max_{a'} Q(s',a')]$
```

In the above, α is the learning rate, $0 \leq \alpha \leq 1$. It represents the balance of retention of past experience and the significance of new data. The process converges for any value of the parameters γ and α, but the speed of convergence depends on these values, hence it is important to adjust them appropriately.

Besides slow convergence, which is a feature of Q-Learning itself, problems arise when the adaptation of Q-Learning to a particular domain is short of meeting the requirements of providing an ideal MDP. Significantly, the *perceptual aliasing* problem is the poor reliability with which the robot is able to identify which state it is in [4].

Algorithms based on artificial neural networks [8] can be used in various forms for learning. However, we prefer a method such as Q-Learning for its greater introspective amenability (it seems to allow us to more readily answer questions such as “why was this particular action taken in a given situation?”—if enough history is kept[5]; neural networks tend to alias effects, making it difficult to discover the influence of a particular effect on the final weights in the system.

1.3 Preceeding behaviour-based work on the EyeBot platform

We have reviewed the dissertation of Tom Walker’s Final Year Project (2006) [1], in which he developed behaviour-based controllers for navigation tasks with different arrangements of obstacles. Machine learning was used in the rotation of behaviours[6], and was based on instantaneous PSD sensor readings (which, quantised, defined the state-space for Q-Learning).

Paying particular attention to section 6.6, in which paths are plotted and analysed, we see that while generally successful, even the best controller encounters a problem in a box canyon scenario, where its path makes several loops in one area before escaping and proceeding to the goal. This highlights the importance of being able to identify, as a developer, unfavourable conditions and to modify behaviour just for those conditions, while not changing the successful behaviour elsewhere.

In section 7.3 (Future Work), Tom recommends that an expanded set of behaviours would benefit the effectiveness of the controllers and that a graphical user interface would simplify the creation of behaviour-based applications. He also considers that the addition of a high-level deliberative planner would add to the robustness of the platform.

[5] The answer would be: “because it received the rewards $R_1, R_2, R_3, ...$ in particular situations.” This allows insights to be gained and the algorithm to be modified accordingly.

[6] Note that in this work we use Q-Learning also, but in a very different mode—for the learning of trajectories.

1.4 Direction of the project

While reviewing the literature, we constantly returned to the question:

> Why do we not have advanced service robots in operation today?[7]

It is a fact that the computer vision means available to us in the present are modest, but even if we had a system capable of reliably estimating 3D structure, recognising objects and tracking moving features, it still would not be a trivial task to make use of this information for controlling a robot.

Our high-level aims are to:

1. Streamline the process of editing behaviour selection rules.
2. Facilitate the implementation of some behaviours as learning modules.
3. Provide sufficient introspection into both behaviour selection and learning modules to enable useful conclusions to be drawn, which can be used to improve the system.

Our system shall be parallel-reactive, with any iterative features encapsulated as opaque agents.[8] It shall allow for reflective and higher-order behaviours by sharing information internally, similarly to a blackboard. Any such shared information should be very well defined.

Learners should be as simple as possible, one learner attached to one type of movement. Such an approach to learning would keep learners' state spaces to a manageable size, making learning efficient.

[7] There are commercially successful service robots, such as the behaviour-based vacuum cleaner Roomba, with sales reaching several million units in the US [24], but they are very simple and will not scale to advanced tasks, such as cooking, without major advances in sensing technology and robot skill development technology.

[8] The actual prototype implementation is sequential, but only to simplify the prototyping process. It is parallelisable and the parallelisation is the next logical step after the system is deemed satisfactorily stable.

Chapter 2

Multimodal behaviour learning

We first analyse the requirements of the system we have sketched in the previous section definitively, and then we describe the real prototype that we have created.

2.1 Requirements

As was mentioned in the Introduction (page 8), we deal with tasks which often have a well-defined high-level hierarchy of goals and sub-goals. One major requirement, therefore, is the ability to track the situation in terms of the current goal, the top-most goal and the goals inbetween. In other words, a planning system is most appropriate at this level. The nature of these goals is the same as the nature of directions on a culinary recipe—unambiguous, essential steps on the way to accomplish a task. For example:

1. Pick up container.
2. Move the container over the bucket.
3. Tip the container over.
4. Return the container to its original location.

There can be no alternative way to accomplish the task, although many things might go wrong. These procedures may be effectively represented by one or more finite state machines.

The identification of the state of affairs itself can be a challenge (in other words, building a system which does only sensing). Such a system must almost never make mistakes—otherwise the robot will be trying to accomplish the wrong goals at the wrong time.

Requirement R1: ability to deal with a well-defined high-level workflow.

Our next requirement is the ability for a system to accomplish goals which are not decomposable, in the sense of the foregoing discussion. Examples are:

- Moving an object while avoiding other objects (on a surface or through space, with a manipulator).
- Grasping objects of different shape securely.
- Opening a door.
- Opening a sealed box.
- Straightening out an arbitrarily-folded sheet of textile.
- Cutting a variety of materials with a blade.
- Connecting together components such as constructor set blocks.

Requirement R2: ability to learn and perform single-purpose movements dependent on sensory feedback and dealing with many degrees of freedom.

Another important requirement which overlaps with the previous two, but is important in its own right, is a vision system segmenting the observable environment into edges and surfaces, permitting the identification and tracking of objects and inferring the 3D position of features.

Requirement R3: a well-developed vision system.

Now, various results of computer vision research make R3 a matter of implementation.[1] We therefore assume that sufficient vision primitives are available to work with R1 and R2. R1 heavily depends on the availability of such a vision system for its most difficult aspect, but is otherwise simple to satisfy. We, therefore, concentrate on R2.

The examples given above for R2 are all composite movements, which may be said to consist of several, in some cases many, phases, without a strict ordering connecting them. There may be multiple equally-effective (on average) ways to proceed when navigating. Force feedback is necessary to judge how securely an object is being held, but it is affected by external forces on the object (for example, if manipulating a drill). In any one of these examples, there is an overwhelming number of factors and special cases, making rigorous analysis and design a major undertaking. Human beings find these examples intuitive.

Much in the R2 examples is reminiscent of feedback control systems, often several acting in parallel. The feedback control systems arise and disperse dynamically, as the process passes through phases (or states). In the rest of the report, we address a way of building systems with the capability to support R2-like processes and their acquisition.

2.1.1 Target applications

We have developed our approach against the application to mobile robot navigation. This application belongs to the class described by R2, and was chosen because it was most appropriate for the hardware and software available to us.

Application A1: navigation through an unspecified warehouse-like environment.

By "warehouse-like" we refer to the restrictions that:

- The floor is flat; and
- The obstacles have approximately vertical walls.

[1] Likely to be enormously time-consuming.

These restrictions are in place primarily to reduce the complexity of the vision system that has to be developed. An additional restriction we introduce is that the floor texture should not have sharp variations in luminosity (lines drawn on the floor, or texture containing clearly defined edges). This restriction does not invalidate our approach, because a stronger vision algorithm, which we intended to implement initially, but then moved out of the scope of the project, would have been capable of correctly interpreting any floor texture.

Robot navigation is the most important functionality of consumer-grade service robots today—from an engineering point of view, all such robots are navigation machines which perform an activity, such as vacuum cleaning, or lawn mowing, as a side effect [20].

Application A2: pushing an object through a warehouse-like environment.

While this task may not have many directly useful applications, it is similar to some of the useful tasks that a robotic manipulator would be required to perform, and it seems to be sufficiently complex. While our system is geared to begin to learn for this task, we have not been able to train robots for it, as the EyeSim simulator does not, at present, facilitate a collision algorithm which is realistic enough for training to have meaning in the real world.[2]

The aim of the project is not to solve these particular problems—it is rather to develop a method that would allow these, and similar problems, to be solved simply.

2.2 Features

We now proceed to formalise the features that our system needs to have in order to address our core requirement R2.

[2] Objects are moved upon colliding with a robot in a way which contradicts the laws of physics severely. While originally planning to upgrade the collision engine in EyeSim, or at least to implement an object as a masquerading robot which would obey the proper physics of collision, time constraints have prevented us from doing so.

Interaction with a high-level planner

A high-level planner (R1) will not be a part of our prototype—at this stage we concentrate on achieving a single goal only (warehouse navigation), from a hypothetical "recipe" of necessary goals (see page 23). The purpose of a high-level planner is to rotate the goals of this type as they are completed. For each goal it would invoke a corresponding *goal-module*. The purpose of the term "goal-module" is to encapsulate the functionality described in R2. A goal-module has a single goal.

As long as goal-modules report the success or failure of achieving their goal reliably, the high-level planner would be a trivial part of the system. Besides rotating goal-modules, it may also need to monitor the environment independently, to respond to emergencies or to revise its strategy when new opportunities become available. A parallel-reactive system, of the type described in [6] (the Cerebus project) would be suitable, because it is able to both support the rotation of goal-modules, and to re-evaluate at a high frequency, which makes it responsive.

While we do not build a high-level planner, our goal-modules must have an appropriate interface to such a planner, to fit into a complete system.

F1. Interface to a high-level planner.

F1a. A goal-module must inform the planner when it has attained the goal.

F1b. A goal-module must inform the planner when it has failed to achieve a goal. (For example, if the robot is stuck and cannot proceed, or if a time limit has been exceeded.)

Activity phases or interim goals

As discussed in the Introduction (page 8), the activity of a single goal-module, as it proceeds towards its goal, is partitioned temporally into identifiable non-reducible phases, or trajectories. The overall trajectory may be influenced by certain constraints for a length of time, and later it may be influenced more by another set of constraints. These patterns are the phenomena which behaviours were first introduced to encapsulate.

We may say that a behaviour corresponds to one *interim goal* (such as "move away from that obstacle") and the degree to which this behaviour is in control of the system is the degree to which the corresponding interim goal is active. (We may have a few behaviour cooperatively controlling the robot, as with a vector-sum arbiter—then the analogy is that the system is pursuing several interim goals at the same time.)

F2. A goal-module is a set of distinct behaviours.

F2a. Each behaviour is active under definite conditions (called the *support* of a behaviour—a subset of state in the state space, to be explained later).

F2b. Each behaviour corresponds to one interim goal.

F2c. A behaviour should be as simple as possible. (If a behaviour could be decomposed into two, it should be.)

F2d. A behaviour combines with other behaviours active under the same conditions (i.e. they have overlapping supports) in a simple, specified way: adding to, overriding the other behaviours, or through the system making an arbitrary choice from among the active behaviours.

For F2a, we could add that the support for each behaviour should be maximally wide, rather than having many similar behaviours differentiated by unnecessary conditions. This is a heuristic for the operator to be aware of when they edit the behaviours.

State space size and state monitors

We would like to make use of all the available sensors, and of memory as required. Our state space, if we consider all possible values of the sensor outputs and the memory, would therefore be very large. Behaviours would need to make use of monitors to define their support. A monitor is an implementation of a function over the state space which monitors a relevant feature of the state (e.g. the density of obstacles in a given direction). There is potentially a many-to-many relation between behaviours and monitors, such that many behaviours could rely on a certain monitor, and likewise the converse.

F3. Monitors extract important features from the state.

F3a. Monitors maintain any memory they need.

F3b. Monitors may use other monitors' outputs.

The design of a system's monitors is highly application-dependent and is likely to be non-trivial. A monitor would only be added if it is required, hence maintaining the memory within the monitors goes some way to ensure that memory is handled efficiently (that "information" is stored rather than "data", roughly speaking).

The human operator's expertise

Because many motion-based tasks are difficult to describe, but are intuitively known by the operator (i.e. motor skills, in the psychological sense), the process of developing behaviours should be guided by an operator controlling the robot.

F4. An operator controls the robot to accomplish the goal of the goal-module. The operator then identifies the distinct behaviours (or interim goals) that were used in the process and specifies each in terms of the salient monitors and the interim goal. This is done iteratively.

Learning the behaviours

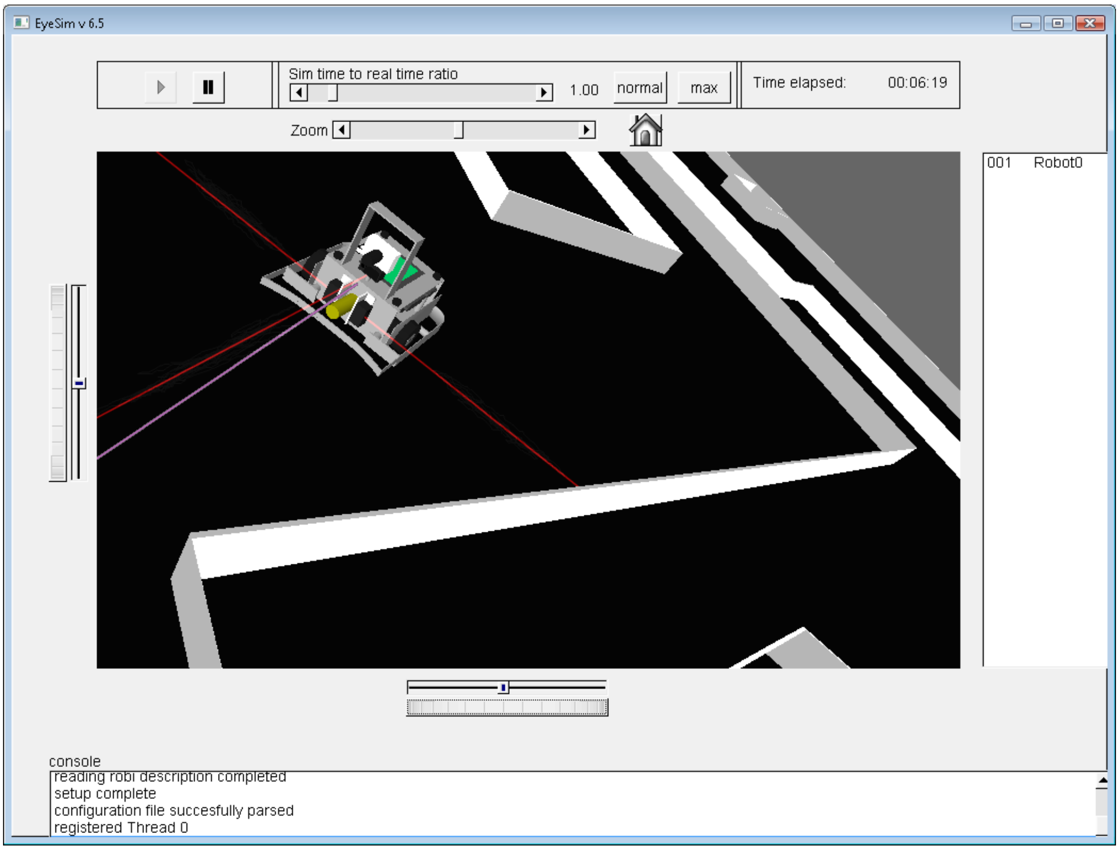
Complex motions which are not readily specified in code, must be learnt.

F5. When the operator, and then the system autonomously, controls the robot, as a behaviour with a learner attached is selected, the learner updates, over time optimising the motion it is responsible for.

Our fundamental design objectives are the systematic introspection into behaviour cycling in real time and the minimisation of the delay between making a change and testing its effects. These objectives guide how the user interface should display the monitors, the behaviours and the sensory data received by the robot. Best results would be achieved when the operator can see what the robot sees—relying only on the same view of the world as the robot itself.

2.3 Description of the target hardware

We work with a simulated model of a real robot, using only the simulated capabilities corresponding to those of the real robot. Figure 2.1 shows the simulator view of the robot model.



```
reading robi description completed
setup complete
configuration file succesfully parsed
registered Thread 0
```

Figure 2.1: EyeSim simulator 3D view of the robot inside a “warehouse-like” environment. The camera is angled slightly downwards to capture nearby obstacles. The directions of the three PSD sensor beams can be seen.

2.3.1 Portability of the code

The design of the EyeBot hardware platform and the EyeSim simulator is such that it allows the same source code to be used, without modifications, on both platforms. Only the compiler has to be switched.

Our GUI works only on a PC (tested only under Windows), and the corresponding code is hence isolated with "`#ifdef EYESIM`" statements. The portable parts of the system have no knowledge of the GUI. The GUI queries them to display their internal state. A full diagram of the classes appears in figure 2.2.

Our proof-of-concept implementation is highly sequential. Monitors are updated first, in order, then the action rules are evaluated. Then the process repeats. The model, however, allows for a high degree of parallelism: independent monitors may be updated simultaneously, or at least, on separate threads. This would be necessary for very high responsiveness in practice, because the sequential implementation causes time-consuming monitor update routines to become bottlenecks. We had the option of a sequential implementation and have taken it for its greater simplicity—making our design process more efficient. Later the algorithms could be parallelised and otherwise optimised to create a production prototype.

2.4 Description of software

We describe the algorithms used together with the design of the user interface of our system. The system consists of approximately 10 000 lines of code, not including the simulator.

The main window is the `NavAssistWindow`, shown with annotations in figure 2.3. A description of the user interface elements follows.

- **Spatial map.** Displays the local map built from sensor data. A thorough description is provided below.[3]

[3] Note: there is a rarely-occurring bug where the spatial map becomes unresponsive (is no

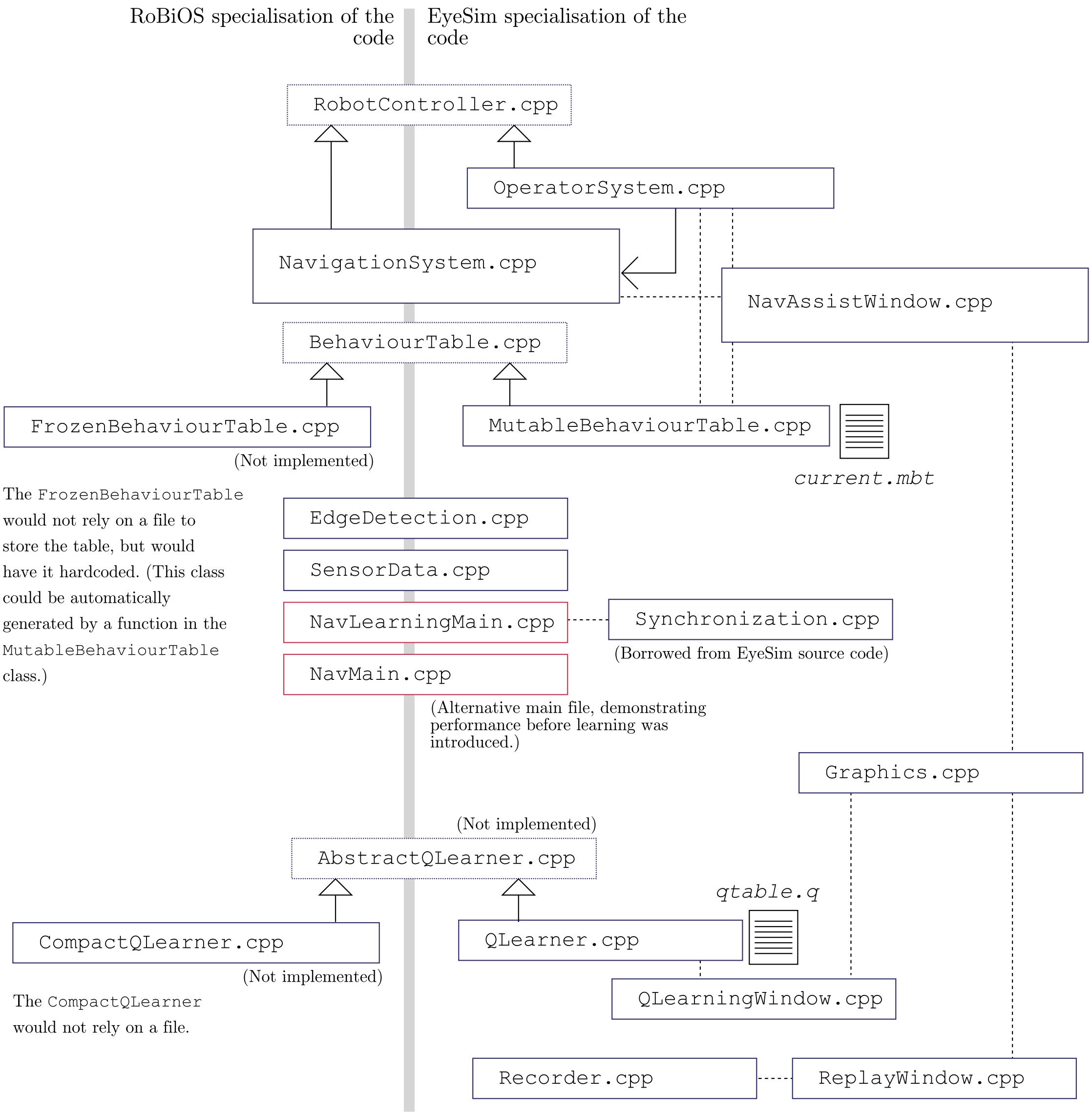
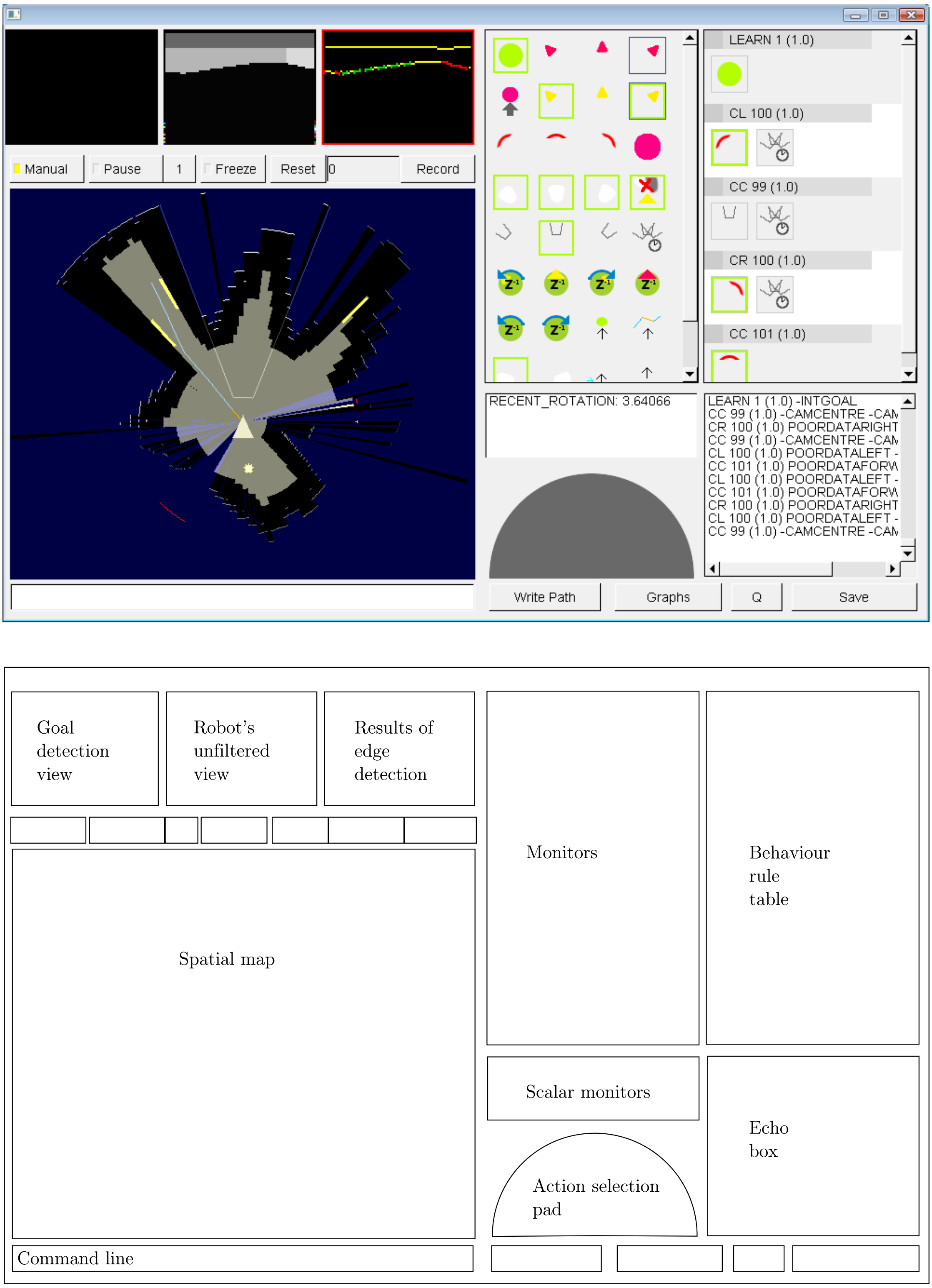

RoBiOS specialisation of the code
EyeSim specialisation of the code
RobotController.cpp
OperatorSystem.cpp
NavigationSystem.cpp
NavAssistWindow.cpp
BehaviourTable.cpp
FrozenBehaviourTable.cpp
(Not implemented)
MutableBehaviourTable.cpp
current.mbt
The FrozenBehaviourTable would not rely on a file to store the table, but would have it hardcoded. (This class could be automatically generated by a function in the MutableBehaviourTable class.)
EdgeDetection.cpp
SensorData.cpp
NavLearningMain.cpp
Synchronization.cpp
(Borrowed from EyeSim source code)
NavMain.cpp
(Alternative main file, demonstrating performance before learning was introduced.)
Graphics.cpp
(Not implemented)
AbstractQLearner.cpp
qtable.q
CompactQLearner.cpp
QLearner.cpp
(Not implemented)
QLearningWindow.cpp
The CompactQLearner would not rely on a file.
Recorder.cpp
ReplayWindow.cpp

Figure 2.2: The implementation of the system, showing the most important relationships between files. The files containing the `main(..)` function are shown in red (two versions). Abstract classes are distinguished by dotted boundaries. Classes entirely on the right of the grey line are EyeSim-only, while the classes that are entirely on the left are intended for RoBiOS only, although they may be tested on the simulator also. The classes appearing across the grey line are the same for both platforms.



LEARN 1 (1.0)
CL 100 (1.0)
CC 99 (1.0)
CR 100 (1.0)
CC 101 (1.0)

RECENT_ROTATION: 3.64066

```
LEARN 1 (1.0) -INTGOAL
CC 99 (1.0) -CAMCENTRE -CAN
CR 100 (1.0) POORDATARIGHT
CC 99 (1.0) -CAMCENTRE -CAN
CL 100 (1.0) POORDATALEFT -
CC 101 (1.0) POORDATAFORW
CL 100 (1.0) POORDATALEFT -
CC 101 (1.0) POORDATAFORW
CR 100 (1.0) POORDATARIGHT
CL 100 (1.0) POORDATALEFT -
CC 99 (1.0) -CAMCENTRE -CAN
```

Figure 2.3: The main window of our prototype, with labels naming elements of the user interface.

- **Goal detection view.** Displays the segmented goal when visible by the robot. The goal is a generic tall red object.

- **Robot's unfiltered view.** Displays the frame as read from the (simulated) camera. Note that this will change unexpectedly as the robot will activate behaviour rules to scan the environment.

- **Results of edge detection.** Shows the image after all filtering steps, which is used to derive obstacle boundaries.

- **Monitors.** Each icon represents one monitor. A green frame means that the condition watched by the monitor is present. The absense of a green frame indicates that the condition is not satisfied. Monitors may be selected by the user by clicking and Shift-clicking to select multiple monitors. Selected monitors appear with black outer frames. Two monitors in the upper right corner of the monitors box are shown selected in figure 2.3. The selection functionality is available for the purpose of defining new behaviours, described in a later section.

- **Behaviour rule table.** Each behaviour rule is a panel. The user may select panels by clicking and then use arrow keys to move rules up or down. This ordering has no effect on how behaviours are chosen, but logically related behaviours should be grouped together. The support of each behaviour rule is shown as the monitor icons (a behaviour rule is applicable if the monitors in its support all have a state matching that displayed on its panel. Applicable behaviour rules are shown with a light blue square. Competing behaviour rules are shown with a dark blue square, and the selected rule is shown with an orange square. The relative weight of competing behaviours is shown as a yellow bar: the size of the yellow bar as a fraction of the grey bar underneath is the probability that the behaviour rule will be selected. The label of a behaviour rule displays the *action* key, the level and the current

longer refreshed) after some running time. Data should be saved and the application should then be restarted.

weight in parentheses. If the weight is represented by a weight function, an equal sign is shown before the calculated weight. (Note: tooltips include the weight function definition.)

- **Scalar monitors.** Displays the names and values of scalar monitors.

- **Action selection pad.** An alternative means for the user to control the robot, required for learning: the user clicks and holds the mouse pointer somewhere over the "pad" to set the velocity of the robot. Distance from the bottom, centre of the "pad" increases with increasing forward velocity. The angle subtended by the bottom edge and the line from the bottom, centre point to the cursor is related to the rotational velocity (omega) which is set: it is zero for the vertical axis of the "pad" and maximum clockwise and anticlockwise at the extremes. A more intuitive method of controlling the robot is through the use of the arrow keys on the keyboard (but this is not suitable for learning). Note that manual mode must be on for either direct control method to work.

- **Echo box.** Displays the selected behaviour rules and feedback for commands entered using the command line.

- **Manual button.** Overrides the navigation system's control of the robot motion. The operator is then able to control the robot using arrow keys or the action selection pad.

- **Pause button.** Prevents rules from being sent to the navigation system. Useful for debugging behaviour rules.

- **The "1" button.** While paused, allow one rule to be sent to the navigation system. Useful for debugging.

- **Freeze button.** Prevents the navigation system from doing any processing. This is used when replaying a recorded session.

- **Reset button.** Resets the state to the first recorded frame.

- **Recorded frame counter.** Increments during recording. Recording is a memory-consuming process and should not be left running for too long without need.
- **Record button.** Press to start or to end recording.
- **Write Path button.** After typing the a filename into the command line, press this button to write an SVG file containing a diagram of the path traversed during the recorded period.
- **Graphs button.** Press to display the Replayer window.
- **Q button.** Press to display the Q-Learning visualisation window.
- **Save button.** Press to write the current behaviour table to disk. **Warning: this button must be pressed before exiting, for the changes to be saved.**

2.4.1 The spatial map

A spatial map is implemented using a radial idiom: it measures the distance to the closest obstacle in the direction of each degree (360 integers are used for the base map). White pixels indicate the boundaries derived from edge detection (via a simple trigonometric relationship, see the code). Orange pixels indicate boundaries obtained from PSD sensor data. The field of view of the camera (intersecting with the floor) is shown as a white outline. The robot is a white triangle at the centre of the map. A "thread", with light blue or tea-coloured segments indicates the recent path of the robot. Red arcs show the directions which are deemed to be insufficiently known and must be sensed before the robot would move there. The grey area shows areas which are deemed to be accessible (i.e. enough sensing has been done and there are no obstacles in such areas; the centre of the robot could safely move onto any grey location. If the interim goal is a point location, it appears as a yellow star. If it is a rotation to a particular heading, it appears as a double green arrow. If the goal direction is known, it appears as a double red arrow, and if the goal point location is known, it appears as a red star.

As the robot rotates, boundary points that have been visible, but subsequently moved out of the field of view, are retained. Their lifetime is discounted by the amount of motion on the spatial map (the more the robot moves, the more outdated they become and could be a source of error).

The robot does not see beyond the “awareness horizon”, which is set at 2m away from the robot.

The visualisation is reminiscent of those that can often be seen accompanying descriptions of deliberative planners. It therefore seems important to stress that this is not a visualisation of a world model being built, even though it may look like one to the observer. **There is no attempt in this system to distinguish features of the world, such as straight-line walls**—we observe such features in the visualisation, because they are present in the world being sensed by the robot.

The visualisation was essential to guide the development of the algorithms that provide the spatial awareness monitors. However, the algorithm itself uses only seven arrays of integers, of 360 elements each, for all its processing. It **transforms** the sensed data into a form that is amenable to use by behaviours. It keeps a local memory of measurements while they are reliable—also within the same seven arrays. The transformation can be optimised to be extremely efficient, such that the processing bottleneck would be the acquisition of the image frame.

The visualisation is a relatively computationally-intensive process, but it would not be a feature of a production system. Its purpose is to reveal the workings of the transformation algorithms for the developer and for the operator.

Essentially, the transformation process gains us two valuable abilities. Firstly, it allows us to fuse information from the PSD sensors and the camera, by using a representation that is commonly functional for both types of measurements. Secondly, it is a representation in which memory is meaningful—that is, information in this representation remains invariant under known conditions, and may be used while these conditions continue to hold.

Areas of interest

During the development of the navigation behaviours, it was found that a heuristic was required to be able to distinguish areas on the spatial map which could lead to new corridors, from areas which, based on the available data, could not. Such a heuristic was found: accessible areas close to large discontinuities in the boundary map often meant that the discontinuities, when explored, turned into open areas (this is clear from common sense by looking at the robot's view or the spatial map). Such areas we call *areas of interest*. They are marked as yellow areas on the spatial map. Navigation with the behaviours that have been developed almost always prioritises areas of interest higher than other accessible areas, when areas of interest are visible and when the navigation system chooses a new interim goal.

The "thread" trailing the robot, which is used for a number of monitors, is tea-coloured in places from which no areas of interest were visible when the robot passed there (otherwise it is light blue). The lack of areas of interest was found to often, but not always, be associated with a dead-end.

2.4.2 The command line

The command line is multi-purpose:

- Typing in any of the *action* names (see below) in manual mode results in that action being performed.
- If a behaviour rule panel is clicked, typing two numbers on the command line and pressing enter causes the level and the weight of the behaviour to be updated, in respective order.
- Most importantly, to define a new behaviour rule, the monitors for its support need to be selected. They must be in the right state (alternatively

the `current.mbt` file may be edited between program invocations). The commands are of the following format:

`~ACTION 3 10.0; Comment`
Defines a behaviour rule in terms of *ACTION*, level 3 and weight 10.0. The comment describes the purpose of the behaviour rule.

`~ACTION 3 =SCALAR-MONITOR-NAME:3@1,5@2; Comment`
Defines a similar behaviour rule, except that its weight is given as a weight function in terms of a scalar monitor, discussed below.

2.4.3 Design of the monitors

Specific rather than general

An important consideration is that there was no effort at all to make the monitors generic across many different types of robots. Quite to the contrary, the monitors should be designed for a specific model of robot only. This allows us to take full advantage of our knowledge of the robot's features and frees us from the complications of working with multiple levels of abstraction that generic monitors would need.

This means, of course, that monitors would have to be rewritten, or adapted, for each new robot model. The justification is that monitors are (or should be) comparatively lightweight and simple to write.

We make this argument on the basis of general recommendations of R. A. Brooks [21], who suggests that embodied entities use the most readily available shortcuts to sense the environment. In particular, the distance to an obstacle seen by a robot's camera is a simple function of the height of the obstacle's base in the image (for a specific robot with a constant viewpoint height, in an environment with flat floors). So, simple thresholding on this height is enough to find if the obstacle is too close [21]. We use a related approach for our obstacle detection.

List of monitors

●	Pursuing an interim goal (motion: moving or turning).
◤	The goal is visible on the left.
▲	The goal is visible straight ahead.
◥	The goal is visible on the right.
● ⬆	There is a direct path towards the goal.
◤	There is an area of interest to the left.
▲	There is an area of interest directly ahead.
◥	There is an area of interest to the right.
◜	There is not enough data from the left.
⌒	There is not enough data in the forward direction.
◝	There is not enough data from the right.
●	The goal has been reached.
	It is possible to move a short distance forward and left.
	It is possible to move a short distance forward.
	It is possible to move a short distance forward and right.
✗ ▲	Point interim goal cancelled (no longer accessible).
↖	Camera is pointing to the left.
⊔	Camera is aimed forward.
↗	Camera is aimed to the right.
⏱	The camera has been turned very recently.

z^{-1} Previous interim goal was: turn to area of interest on the left.

z^{-1} Previous interim goal was: turn to central area of interest.

z^{-1} Previous interim goal was: turn to area of interest on the right.

z^{-1} Previous interim goal was: proceed to the end goal.

z^{-1} Previous movement-related interim goal was a left rotation.

z^{-1} Previous movement-related interim goal was a right rotation.

● ↑ We are very close to the interim goal.

↑ We are facing a trail which is a probable dead end.

There is some accessibility directly to the left.

There is some accessibility directly right.

→↑ In the recent past our heading changed to the left.

↑ ↑ In the recent past our heading did not change.

↑← In the recent past our heading changed to the right.

2.4.4 List of actions

Behaviour rules are written in terms of generic actions. Neither the behaviour rules or these actions correspond to conventional behaviours directly. Rather, sets of behaviour rules, when taken together, form a consistent behaviour. For example, many behaviour rules which are written in terms of the right side of the robot need mirror rules for the left side; together the right and the left, and possibly the centre behaviour rules, they form a behaviour. The actions listed below are best seen as behaviour building blocks.

MG	Move towards the end goal.
CL	Turn camera to the left.*
CC	Turn camera directly forward.*
CR	Turn camera to the right.*
RL180	Turn around, in the anticlockwise direction.
RL90	Turn 90 degrees to the left.
RLL	Large rotation to the left.
RL	Small rotation to the left.
RR	Small rotation to the right.
RRR	Large rotation to the right.
RR90	Turn 90 degrees to the right.
RR180	Turn around, going in the clockwise direction.
FL	Rotate to face an area of interest to the left.
MC	Move to the centremost area of interest.
FR	Rotate to face the area of interest to the right.
SL	A small step forward and to the left.
SC	A small step directly forward.
SR	A small step forward and to the right.
MM	Pick a random accessible location.
STOP	Stop motion and other actions.
SUSPEND	Suspend motion.*
LEARN	Select a random goal. Set highest learning rate.
-	Continue.*

Entries marked with * indicate non-motional actions for the purposes of setting the very important monitor “Pursuing an interim goal.”

Many of the monitors and actions described have been introduced while using the interface to develop the navigation system. Hence they are, in themselves, relevant as results.

2.4.5 The behaviour selection model

According to F2d, the design of the behaviour selection model has been made flexible. Each behaviour rule must be assigned a level (an integer) and a weight (a non-negative floating point number) at the time of its definition. There are three aspects determining which action is chosen:

1. The supports of all behaviour rules are matched to the monitor states, the matching group being the *applicable rules* for the situation.

2. From the set of applicable rules, the rules occupying the highest level form the set of *competing rules*.

3. An arbitrary rule is selected from the competing rules, such that the probability of selecting a rule is proportional to the weight of that rule.[4]

Levels allow for absolute precedence, when it is needed, while weights can be used to control the relative importance of behaviours when there exist multiple acceptable alternatives for the next interim goal.

Since there are no constraints on the implementation of monitors, any functionality which can be expressed in code may be approximated with a finite number of monitors and behaviour rules—more for a better approximation. The use of scalar monitors and piecewise-linear weight functions improves conciseness substantially in certain cases (see the next section).

2.4.6 Representing behaviour weight variability

When we consider behaviours in the general sense—patterns of motion that a robot might perform in response to an interim goal—we see that often a behaviour's importance, or urgency, varies smoothly with a linear parameter. For example, in basic obstacle avoidance, the importance of moving away from an obstacle increases as the distance to it decreases—smoothly. Or, if a robot depends on a rechareable

[4] The "wheel-of-fortune" method.

battery, the importance of seeking out a recharge station increases with time. In our initial design, we could approximate such a relationship using several behaviour rules and threshold monitors which depend on the linear parameter (e.g. monitors which turn on at $P < 20$, $P < 50$, $P < 100$, etc., where P is the parameter). Then we could have rules such as:

- A: level 1, weight 10, $\not{\texttt{P}}$-LESS-THAN-100.
- A: level 1, weight 20, P-LESS-THAN-100 $\not{\texttt{P}}$-LESS-THAN-50.
- A: level 1, weight 40, P-LESS-THAN-50 $\not{\texttt{P}}$-LESS-THAN-20.
- A: level 1, weight 100, P-LESS-THAN-20.

In the above, A represents the same action. The four behaviour rules really specify one unit of functionality, which may be represented as in figure 2.4.

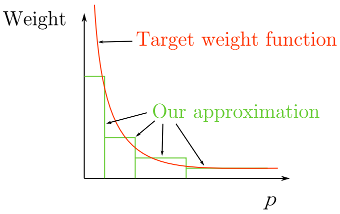


Figure 2.4: How a variable weight would be achieved under our initial, fixed-weight system.

There are two problems with this approach:

1. The approximation is piecewise-constant, not smooth.
2. One conceptual behaviour requires several monitors and several rules to describe, leading to a proliferation of items, reducing clarity.

A solution to this is to incorporate monitors which have scalar values, rather than the binary true/false values of our basic monitors. The weight of one behaviour rule is defined as a function of the scalar monitor. Parameter-weight pairs are supplied and the system linearly interpolates the weight function between these coordinates. Continuing the above example, P becomes a scalar monitor and now only one behaviour rule is required:

- A: level 1, weight=P:130@10,50@30,20@60,10@100

For the pairs we have chosen to use the notation $w@v$, where w is the weight that the function has at $P = v$. The resulting approximation is graphed in figure 2.5. The quality of the approximation is much better than necessary.

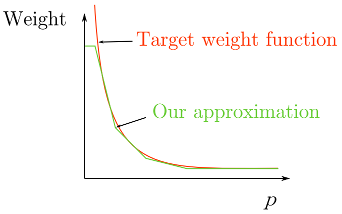


Figure 2.5: How a variable weight would be achieved under our improved weight system.

The scalar monitors can measure any internal or external parameter. There is no reason not to take advantage of a useful scalar aspect of the environment when one is able to write code to detect it. Scalar monitors could be functions of multiple other scalar monitors—this makes the system very general—it is a challenge to concieve of an effect that cannot be expressed using these tools, if it can be expressed in code at all.

The next bottleneck on the way to building a versatile embodied system is indeed about effects which are difficult to express in code—so we use machine learning to capture them.

2.4.7 Recording and replaying

During development, it was found that the system often made decisions too quickly for the operator to take note of the decision and the antecedents. Therefore, a facility was developed allowing the operator to review the progress of the navigation system. Pressing the Graph button brings up the review window (figure 2.6). Pressing the Record button in the main window begins recording.

The facility is useful when identifying the causes of certain decisions, as it allows to freeze in time the state of affairs leading up to that decision, and to examine the applicable behaviour rules at a glance.

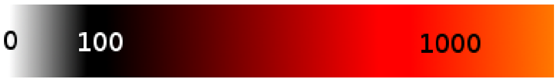
2.4.8 Q-Learning of simple trajectories

Motions involving many degrees of freedom are not strongly present in the navigation task. We used machine learning to learn perhaps the simplest possible trajectory suite: the most efficient driving trajectory to any point location within the field of view of the robot (defined thus such that the robot can see any obstacles).

This value is the measure of the value of the state, and is the most important Q-value for that state. The other Q-values are not shown, but are stored in the file `qtable.q`.

We have set up the Q-Learning algorithm with states being the spatial points on the local map, as shown in figure 2.7. The spacing between the points increases linearly with distance from the robot: the aim was to make movements more precise in close vicinity of the robot. The set of actions is similarly large: the set of all possible V-Omega combinations is spanned by points which are actions. The "Action selection pad" device allows the actions to be sampled in Manual mode.

In the Q-Learning window, the value $\max_a Q(s,a)$ is displayed for each state in colour approximately as shown below.



The "Save Q table" button writes the table to the file `qtable.q`. It is possible to display the next best transition state for each state by clicking the check button.

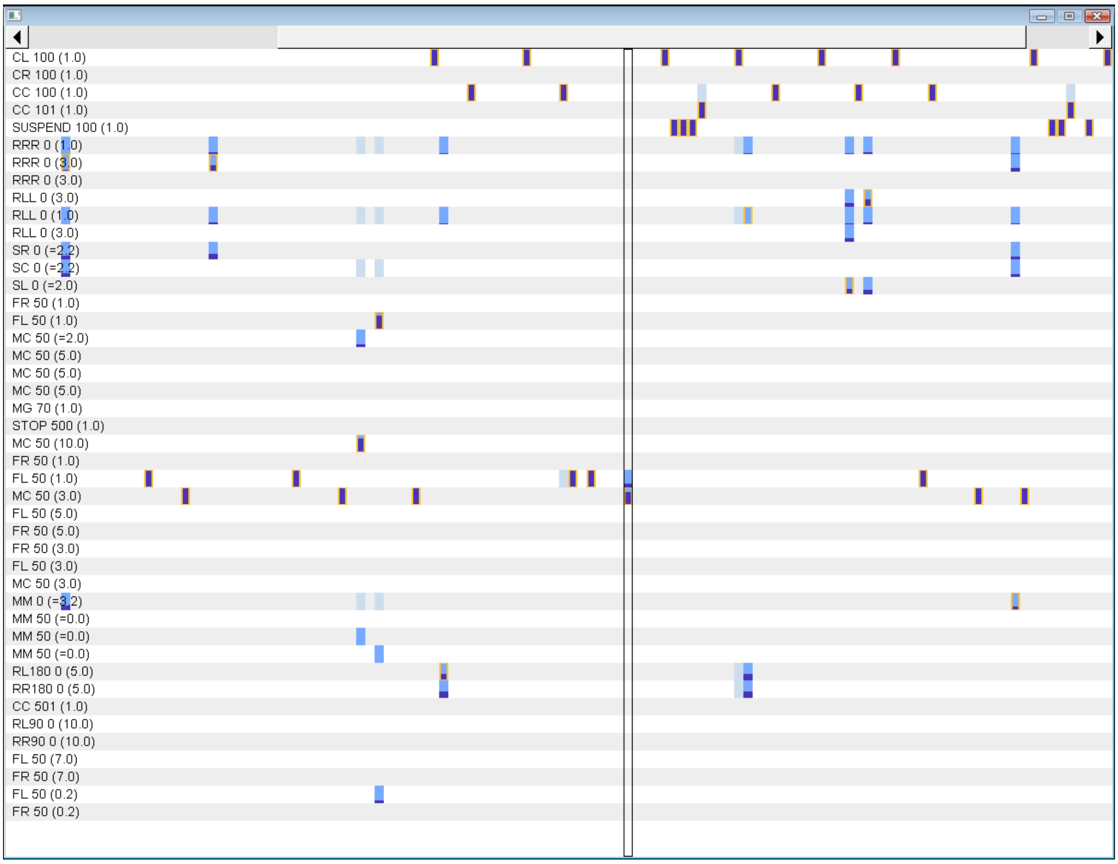


Figure 2.6: The window allowing to replay all recorded states. The rows correspond to behaviour rules—all are shown in the window at the same time. Columns correspond to one saved state of the navigation system. Applicable behaviour rules are shown in light blue. Competing behaviour rules are shown in blue with their current weight as a fraction of the total weight of all competing behaviours superimposed in dark blue. The actual behaviour that was selected at a particular time is shown with an orange frame. Replaying is achieved by dragging the black bar across columns with the mouse. The spatial map is restored to the state corresponding to the column being replayed, as are the goal-seeking area and the area showing the results of edge detection. The states of the monitors are also correctly replayed.

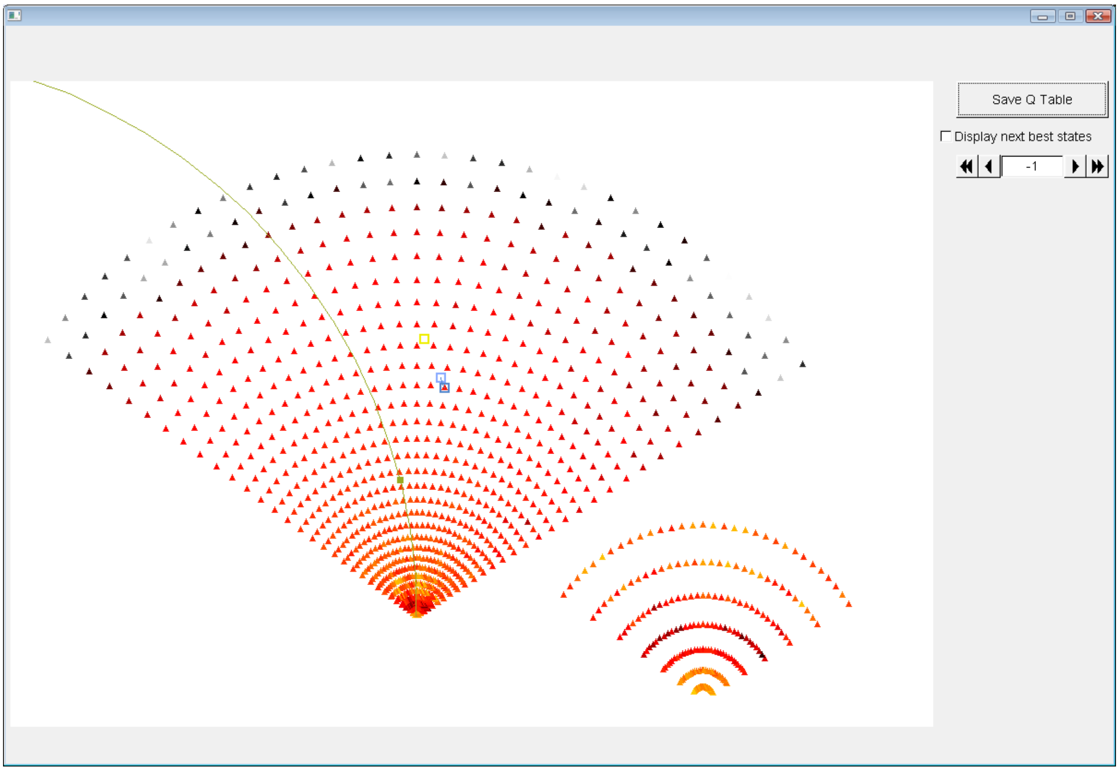


Figure 2.7: The window displaying the state of the Q-Learning algorithm. This window may be brough up using the Q button on the main window. Each state is displayed as it is spatially defined. There is a zoomed-in version of the first several layers of states to the right of the main display.

A line connecting a state and its next best state is drawn for all states where it is known. This helps understand how an exploitation-only strategy would behave. As the algorithm learns, the expected trajectory for the selected action is shown as a green line and the "from" and "to" states are shown as a yellow and a blue boxes respectively.

A special action (in the sense of being a building block for a behaviour, not an action in Q-Learning) called **LEARN** has been added. It sets the exploration-to-exploitation tradeoff ϵ to zero (i.e. maximum exploration). An empty world was created and a behaviour table (called `learning.mbt`) was implemented to scan the field and cycle the **LEARN** action, which selects a random state from the Q-Learning state space and makes it an interim goal. An action selection algorithm is then applied and the (Q-Learning) action chosen is applied to the V-Omega controller for a constant amount of time. The position of the interim goal after the action has been performed determines the state that the system has transitioned to.

Chapter 3

Experimental evaluation

3.1 Development of behaviour rules

A set of 44 behaviour rules has been developed. The rules may be examined from within the program, provided on CD. The tooltips contain the comments describing the purpose of each rule. Generally:

- The camera control behaviours have been placed at level 100, as environment awareness was the most important factor for being able to proceed.

- Rules involving areas of interest (24 rules) have all been placed at level 50, although they have differing supports, which means that only some of them compete at any time.

- Back-up rules, not involving areas of interest, have been placed at level 0.

There is scope for further incremental improvement, but our efforts were redirected to the learning task. We were satisfied that the proposed approach worked as expected. Figure 3.1 contains some representative paths that have been recorded.

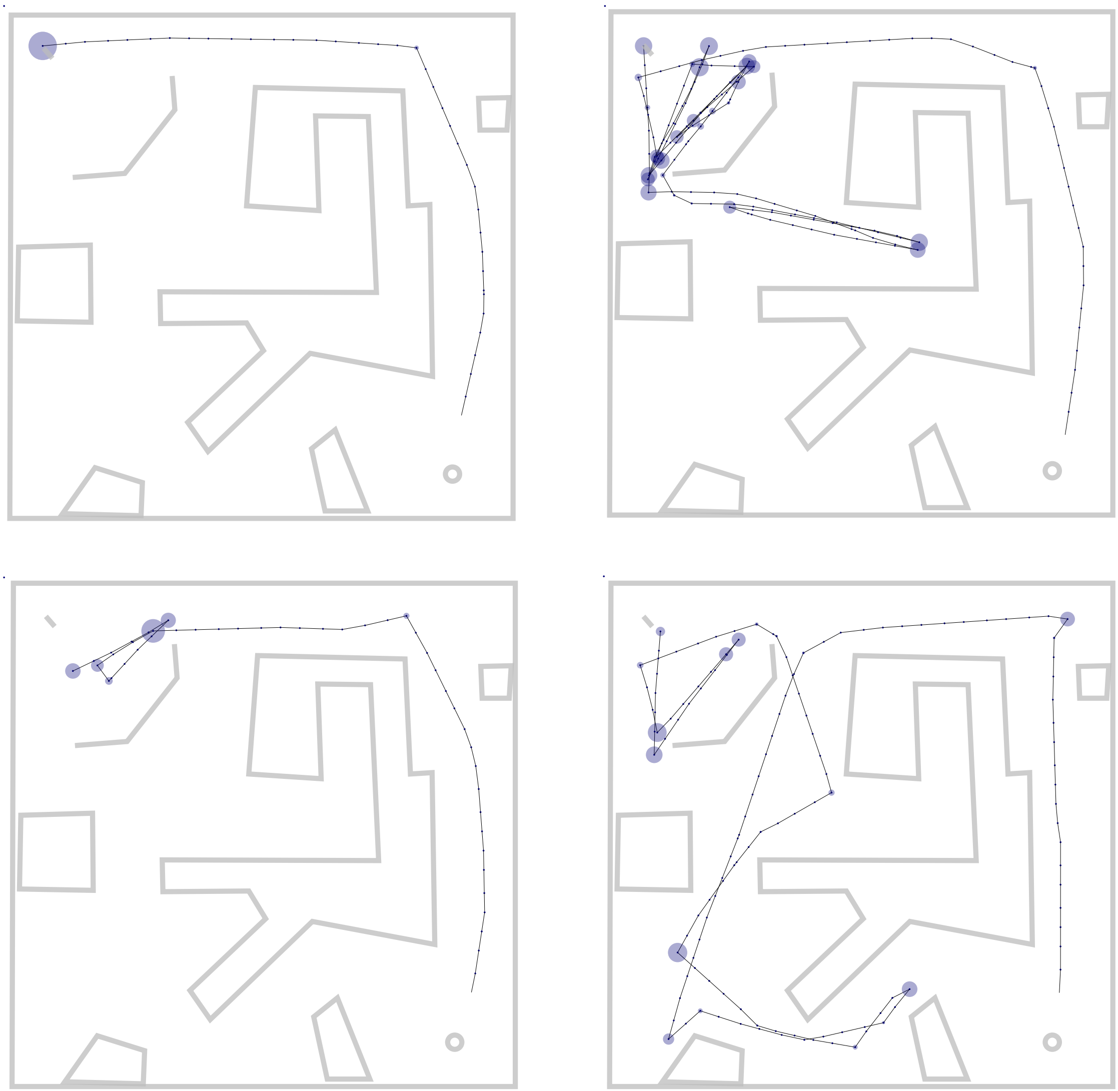


Figure 3.1: Representative paths obtained without the use of Q-Learning (point interim goals were approached by rotating to face them directly and then moving towards them). The circles indicate points of rotation on the spot: the greater the radius, the more rotation occurred. Clockwise from top-left: a frequently occurring path in this scenario: the navigation system is quite good at following corridors; an example showing a problem leaving the initial starting area—this is an example from which further monitors and behaviour rules could be derived to avoid such behaviour; quite sane exploration of the area, except that the robot unfortunately turns around at the entrance to the area containing the goal; another typical trial. Inportantly, the top-right example shows the corrected behaviour of dealing with the dead-end in the middle of the map: the robot turns around upon reaching it.

3.2 Performance of Q-Learning

3.2.1 Modifications to the algorithm

In order to adjust the algorithm to our problem, we have modified the action selection part of it. The conventional selection in Q-Learning either selects the action with the most associated reward, or any action at random. Since we have chosen a very large action space, the random selection of actions would have caused the algorithm not to converge in a reasonable amount of time. In the intensive learning mode, when we set the exploration-exploitation balance to maximum exploration, we now randomly choose one of three selection methods, each with equal probability:

1. We could choose an action which is close to the currently most-profitable action for the present state. This allows the best action to slowly drift towards the optimal action for that state. In our problem, local maxima are also global maxima, hence over time this method alone would have converged to a reasonable policy.
2. We could choose the best action from the most successful (in terms of maximum Q-value) of our neighbouring states. Hence, if a neighbouring state "discovers" a profitable action, it would affect the states next to it. This, and the previous method, are appropriate in our setting, because we expect neighbouring states to have very similar optimal actions.
3. The third method is the original random choice from all possible actions. This allows some states to find good actions by luck.

It was found that after leaving this algorithm to run overnight, at 1:1 simulation time to real time ratio, in intensive learning mode, the average velocity of the actions was still very low (the optimal velocity should be the maximum for all states except the states very near the robot). See figure 3.2. This meant that the algorithm was not converging well.

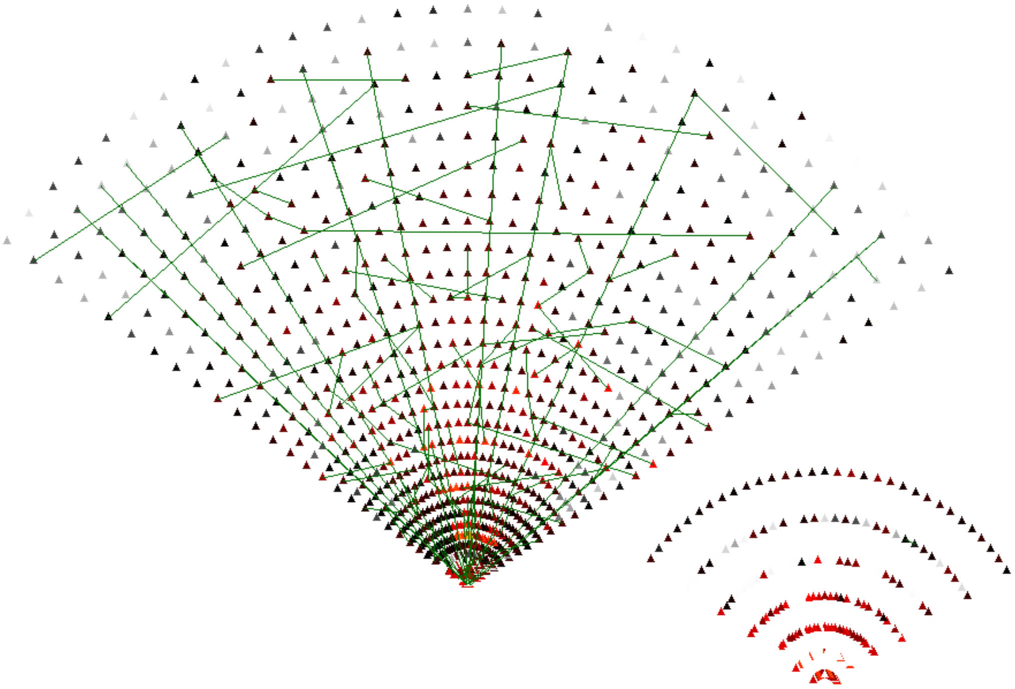


Figure 3.2: The state of the Q-Learning table after the initial training run.

It was then that the “Action selection pad” was implemented. As predicted, the algorithm benefitted greately from the operator’s not always optimal, but often high-quality choices of actions. The algorithm was left for another night after the operator training phase, producing the best of our Q-Learning trajectory controllers (figure 3.3).

Believing that the speed of learning could still be improved, we have implemented a control on the Q-Learning window, which restricts to a particular radial tier the set of states from which the **LEARN** action randomly chooses for intensive learning. It was believed that if the states closer to the robot would be intensively learnt first, and then the radius extended gradually, the overall learning rate would improve. In actual fact, while the Q-values propagated faster, as expected, this strategy produced a policy involving significant unwanted rotation. This may be seen from figure 3.4, from the pattern of the lines.

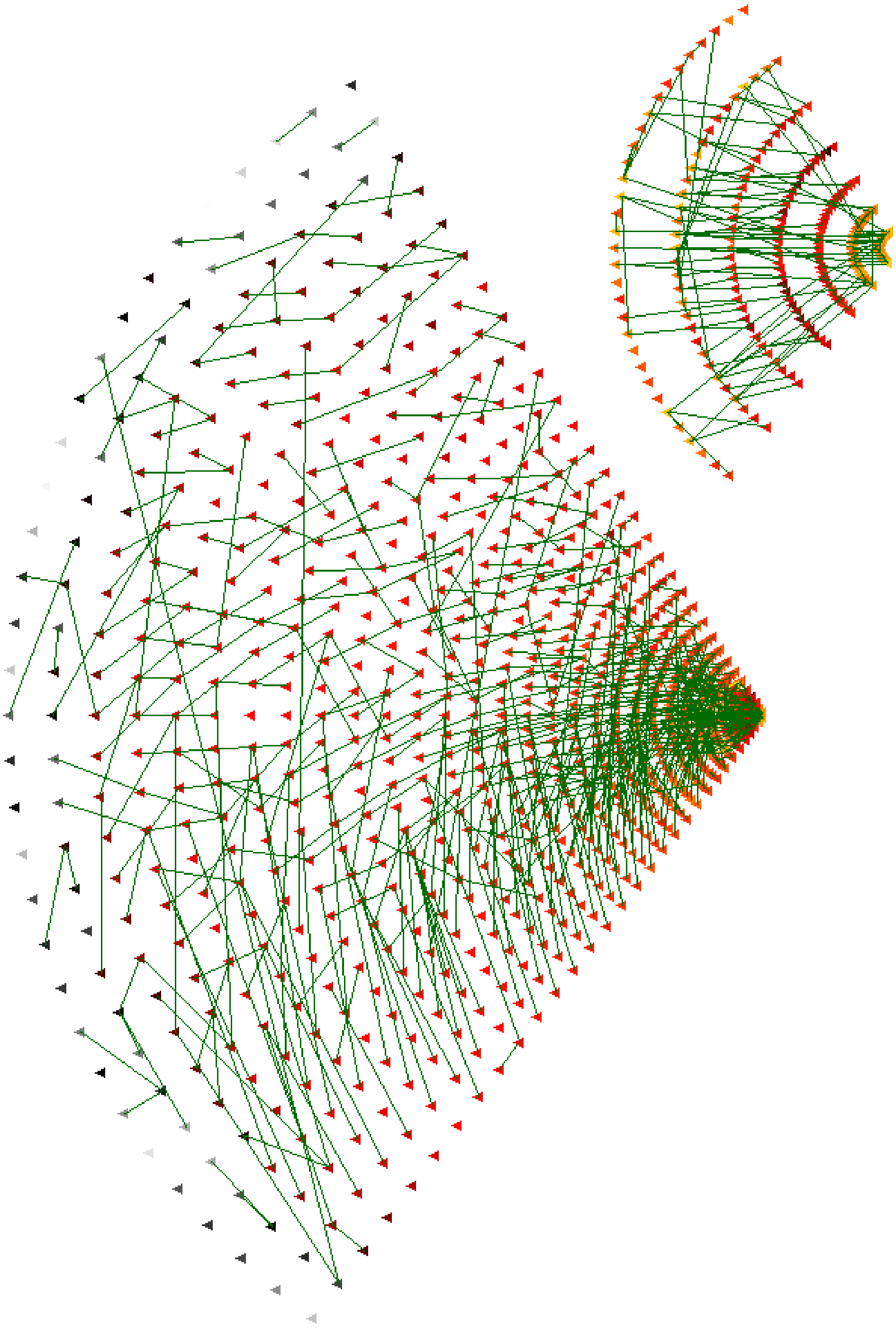


Figure 3.3: The state of the Q-Learning table after training with an operator, followed by an overnight autonomous training.

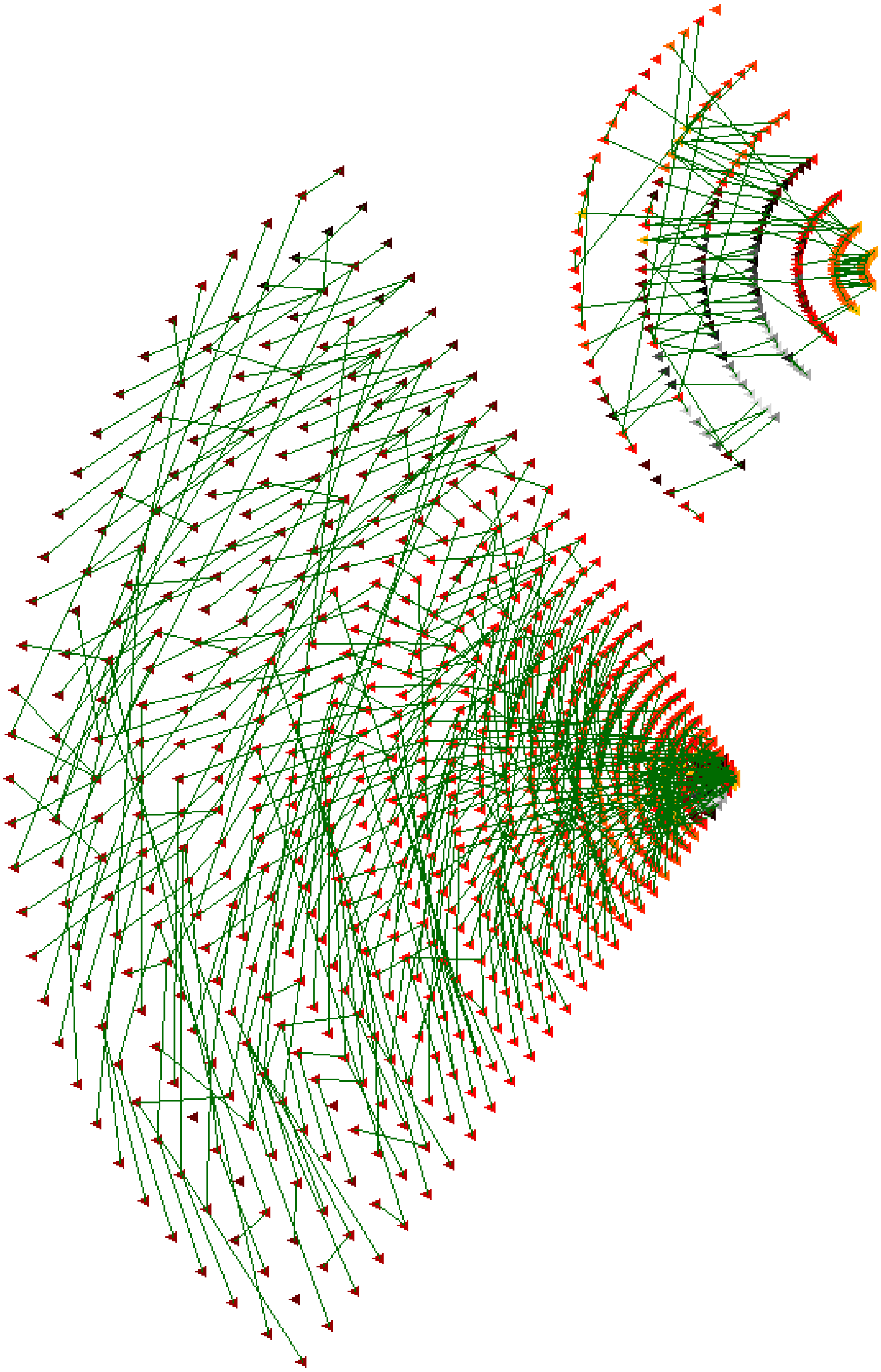


Figure 3.4: The state of the Q-Learning table after “progressive” training.

Figure 3.5 shows several paths that were observed after the Q-Learning module replaced the default cautious driving behaviour.

3.3 Miscellaneous observations

3.3.1 Analysis from a bird's eye view

While we noted in section 2.2 that the operator would be most effective if driving the robot while observing the world through the robot's own sensory means, we neglected the fact that external observation could also be useful. Athletes often record and replay video footage of their performance, to observe it from a third-person perspective and identify aspects which may not be obvious from a first-person view. The internal view is essential, but the more views the operator uses, the more likely they are to detect useful heuristics.

3.4 Review of requirements and features

Feature F4 (describing the operator's strategy in developing behaviours) in practice changes as the behaviour table is built up. At the beginning the author indeed manually traversed the navigation scenario, noting heuristics that needed to be used, but as these heuristics were incorporated into the behaviour table, the system gradually attained acceptable performance levels. The operator's strategy then changed: the operator observes the system in action, identifies weaknesses and augments the behaviour table (patching the system).

Having developed the navigation system, we, as an operator, find our design of the behaviour-formulation workflow to be a workable means of eliciting valuable rules-of-thumb.

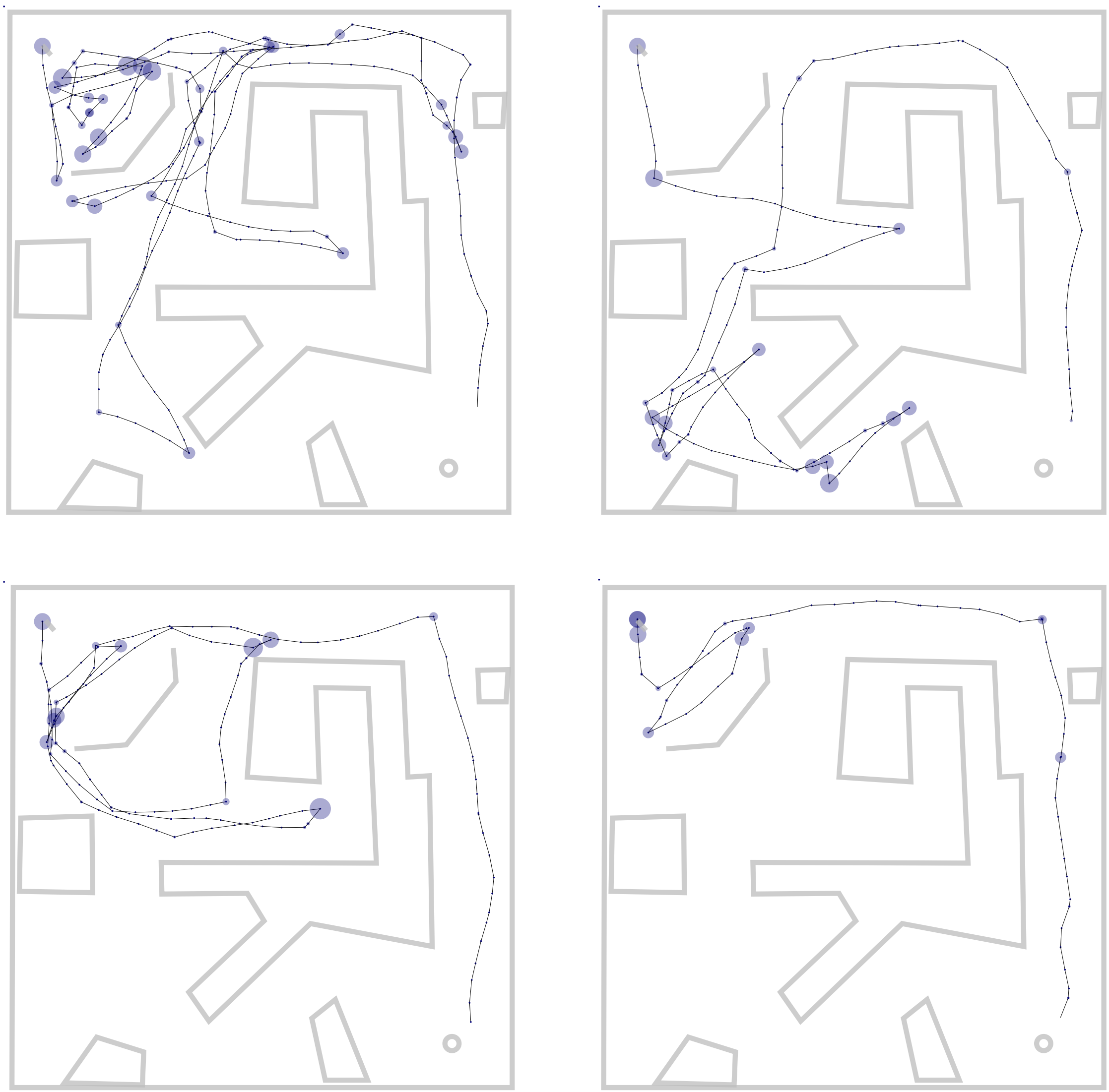


Figure 3.5: The motion towards the interim goal was much faster than with the original implementation—the learner had learnt the near-optimal V-Omega parameters to reach an interim goal, while before the implementation had to set conservative values, in order not to skip past the interim goal point. The camera frame rate seems to be too slow to keep up with the speed of the learner. Hence it was found that the navigation system made rapid decisions based on cached boundary data before it was updated by a new frame, in effect causing the robot to avoid non-existing obstacles and trace oscillating patterns, as in the top-left figure. Again, our visual interface allowed us to identify the problem. It may be easily fixed through the introduction of monitors to delay making decisions until a new frame is processed, for instance. Notwithstanding these problems, the system with learned trajectories could produce paths of good quality, as shown in the diagram.

3.4.1 Why an operator is necessary as a source of heuristics

Consider our experience with developing behaviours for the navigation task. When we have identified the “areas of interest” heuristic (prompting us to explore accessible regions near discontinuities in the visual boundaries map), this was in response to the question “are some accessible areas more attractive for exploration than others?” Our initial direction-selection heurstic was based solely on the reach of the accessibility region—the further it stretched in a certain direction, the more likely we would be to follow that direction. The problem was that sometimes the machine would choose a direction that the operator would never choose under the same circumstances. A most obvious example is, when facing a medium-length corridor with a clear dead-end, the operator would back-track, but the machine would not.

Our question now is: it is within the ability of a machine learning algorithm to derive this heuristic independently? We cannot highlight the importance of border discontinuities to the algorithm, because doing so would defeat the purpose—the algorithm would surely find this heuristic, but *our* main achievement, as an operator, in elucidating this heuristic was to identify that boundary discontinuities are significant.

A machine learning algorithm may be applied by placing the robot at one particular point on the map, and allowing it to test different directions at random, before being returned to the initial location for another trial, in a long series of such trials. We can suppose that it would identify areas closer to the boundary discontinuities as more attractive (because they really are, on average). But this knowledge would be useless in another scenario, because the algorithm would not have linked the increased attractiveness to the proximity of boundary discontinuities. This is because a machine learning algorithm does not “think” in terms of boundary discontinuities and it has no facility to **identify them as a feature**. This is the saliency problem well-known in AI. (Described for example in Chapter 3, “Artificial Intelligence: From High Hopes to Sober Reality,” in [23].)

As an operator, we were able to identify the feature from experience of years of navigating in the real world, but even if we ignored this experience, a boundary discontinuity is a feature that clearly stands out to us—it is provided as "hardware", in some sense, as a result of our evolution. Many features stand out—the lightness of surfaces, the sharpness of corners, and many others beside. This makes it easy for us to correlate one particular feature with advantageous decisions.

A device would need to be provided with the ability to distinguish thousands of features, before it will be able to compete with the human brain at deriving heuristics. Such a device would need to be enormously complex. It seems naïve to think that a simple algorithm would solve this problem. This is why a human operator is an essential component of our system. They are a source of heuristics.

Thus we believe we were justified in placing the responsibility of providing the strategy onto a human operator, rather than setting up a machine learning algorithm to attempt to learn a strategy. It seems that the domain is simply too complex.

3.4.2 The decomposition of functionality into monitors and behaviour rules

A possible criticism of our approach may be that most work during development goes into setting up the monitors. Editing the rules introduces significant changes into the behaviour, but only as a consequence of the rules relying on the monitors. So why have we divided the labour so unevenly?

The defence is that monitors report information **which is easily observable by the operator visually**. It is therefore straightforward to test the correctness of each monitor's implementation. This division of labour also provides crucial insights when things go wrong: the reason why the robot performed a certain action becomes very clear by the examination of the rule that was in control and the values of the monitors at that time. There is no need to guess an explanation.

Another way of looking at monitors is that they are many modules with relatively simple functionality (simple to comprehend) and we are not concerned with their implementation, just that they work reliably. We are, however, very concerned with the way in which simple actions are combined, hence we are interesting in closely examining *that* mechanism. Monitors remaining black boxes help us hide unnecessary complexity.

3.4.3 Editing behaviours versus learning strategies

Considering the navigation task, we could have stopped short of letting the operator edit rules, and limited their role to defining monitors for extracting features (which, in the previous section we argue, is only feasible with the help of an operator). It would be possible to use an ML algorithm to build the equivalent of our behaviour table—to provide appropriate actions, based on the states of the monitors. We have chosen not to do this, because we do not see any significant improvement to be gained by such an algorithm over our best behaviour table. It is likely, provided we were successful in setting up the learner, that the performance of the two policies would be comparable, because there is a straightforward relationship between the states of the monitors and the set of appropriate actions to take—in most cases these are clearly evident. In other words, our best behaviour table represents a strategy that is close to the optimal possible for this task (considering several objective performance metrics we could define), and a learning algorithm would produce just another approximation to the optimal. Moreover, we have no way to guarantee an acceptable convergence time.

3.5 Relationship to similar systems

Our *monitor* may be compared to a perceptual schema as described by Arkin [2]. The idea also appears in the description of the 3T architecture [17].

3.6 Future work

3.6.1 Transferring the system onto real robots

There are a number of differences between the simulated and the real robots. In particular, the characteristics of the camera: the field of view, abberations, focus, etc. Our system would need to be adapted before it can successfully control the real robots—however, the bulk of it should carry over without changes. Only the sensory processing code would be significantly affected.

The most effective way to achieve the transition initially would be to write a communications system which would forward the camera output and PSD sensor readings to a suitably modified version of our operator's software. It would forward the servo settings and the V-Omega parameters back to the robot.[1] It would then be possible to write the necessary extra image filtering procedures, testing them immediately on the live image stream. In this phase of development, the robot is a terminal which does not perform significant processing on-board.

The learners would have to be adjusted on the real robots, through training, to account for the differences in motor characteristics and physical effects not present in the simulation. It is unclear how quickly the simulator-trained learners would be able to adapt to the real world. It is probably best to adjust the simulator's model of a particular robot's dynamics by measuring its real-world response characteristics, and then learn trajectories on the simulator. This would require changes to EyeSim's V-Omega module.

Once the robot-as-a-terminal system performs acceptably, and is ready to be fielded on an untethered, autonomous robot, the system would first have to be optimised. During the construction of our prototype, liberal use was made of space and of processor power, to speed up our development cycle. During optimisation, multiple image filter passes could be combined into one. Not all of a frame would have to

[1] This has been planned for: all code interacting directly with RoBiOS is located in `NavLearningMain.cpp`, and the class `SensorData` represents a packet that would need to be transmitted across the wire.

be processed. (Only up to the first obstacle edge from the bottom of the image). Also:

- Floating-point operations could be replaced by the use of integers (e.g. a fixed exponent idiom), for a gain in efficiency where needed.
- Trigonometric operations could be replaced with look-up tables.

In the latter two optimisations, we would be trading precision (we do not need accuracy to 6 decimal places) for speed.

3.6.2 Mapping out a room

Mobile service robots would often work within the context of a room, or a set of rooms, such as a floor in a multi-storey building. Their applications would very often require them to work in terms of a map of their context. Building such a map manually, for each robot-room pair deployed, may be a considerably tedious task.

We note that our spatial awareness map is currently only being used to determine accessibility, with the data discarded once it leaves the awareness radius. Instead, it could be used to build a persistent map of the territory. Substantial work would be required:

- The drift inherent in the internal position-tracking facility would need to be taken into account. In practice, a reasonable technique might be to locate features (i.e. corners) in the boundary, assume that they are connected with straight walls, discover these features and walls by wall-following, and then to navigate from feature to feature in a random sequence (across rooms and corridors). Each feature-feature trip would provide a measurement of the relative displacement of the two features. Combined with an assumption that most walls are at right angles to each other, it seems plausible that an accurate map could be produced in the end.

- More sophisticated image processing would be necessary, to deal with patterned floors, shadows and varying lighting conditions. Indeed, it is likely that immediate edge-detection would be insufficient. Edges would need to be tracked as the robot moves, iteratively estimating their distance away from the camera. Whilst this approach seems to be useful and transferable to other domains, it would be a matter for a whole separate project. Making heavier use of PSD sensors, or even adding a sonar module, may be a much simpler alternative.

There are three apparent benefits of an ability of a robot to build its own map (besides relieving a human being of this activity):

1. The features that the robot determines will be recorded as seen through its own sensors and sensory processing mechanisms. We expect that it would be significantly easier to match a feature to one that has been previously identified using the same sensors, than to correlate an observed feature with that described on a foreign map.
2. Manual maps are prone to error and must be updated if, for example, large furniture items are moved. A service robot that is able to build its own map could invoke this mode at any time and update the map automatically.
3. The navigation system supplemented with a persistent map could use path-planning to guide its reactive obstacle avoidance. Such a system would appear purposeful and efficient, though it would still be fully reactive (the path-planner would be another set of monitors for the system to consider when navigating). Our present system wanders randomly.

3.6.3 The EyeSim simulator

In the current version of EyeSim, there is a bug which prevents the intended use of the “Sim time to real time ratio” feature, which would ideally allow simulations to proceed at many times the normal speed. As it is, changing the ratio changes the rate at which frames are delivered to the robot, so speeding up the simulation this way affects the robot's perception of the world quite adversely. Hence we have been running our learning algorithms at normal speed. The reason for the occurrence of the bug has yet not been determined. If the bug is fixed, many more variations of learning algorithms could be run, providing empirical comparisons of their effectiveness.

3.7 Conclusion

A minimalist long-term planner sets goals of a persistent nature. Behaviours proceed towards such a goal, reacting to events in the robot's immediate environment. We see one behaviour as reacting to a particular type of event by setting an interim goal (for example: "move to point P"). Certain interim goals require complex motion, and, henceforth, are realised through the use of a trajectory learner.

- A planner is best for ordering and controlling checkpoint goals.
- Behaviours are most appropriate for representing the conditions for, and the precedences of, interim goals.
- Learning is effective at dealing with ad-hoc trajectory control.

We have designed a prototype system to apply these broad principles, to test their practicality. The results have been generally positive, but the navigation task that we have had time to develop is very simple. In particular, we had no practical exposure to effectors with many degrees of freedom, and while it seems plausible that machine learning is the right approach for attacking this problem, practical implementation is the only interesting test.

The process we have devised, of implementing monitors with outputs which can be visually verified for correctness, exceeded our expectations by how well it seemed to work. We have no doubt that the navigation system could be developed further, to incorporate still more heuristics and to achieve superior navigation performance. The process of defining behaviour rules works well with the monitors and we are satisfied with the generality that the combination of both levels and weights offer for our control of behaviour arbitration.

The trajectory learning method performed acceptably, though even after modifications to improve the speed of learning, we still expect that there are much much faster ways to learn using nearly the same system. Such improvements would have permitted many types of trajectories to be learnt for a task such as pushing an object.

In general, having undertaken this study, we believe that the creation of an effective architecture is not a matter of establishing "the right" constraints, but relaxing as many as possible. Instead, a focus should be placed on the introspective power the architecture allows.

In one way, our work may be seen as work on the scaffolding for the efficient editing of behaviours. The real problem being faced in robotics is exactly the kind of inefficiency of breathing life into the hardware that may be addressed by effective human-machine interfaces. After all, the robots are not required to perform anything bewildering—they are required to perform a great number of simple actions, at the right times. It is the efficient management of this complexity that is the bottleneck, and the right user interface seems to be a worthwhile pursuit.

We are convinced that our emphasis on the verifiability of monitors by visual inspection and the traceability of observed behaviours back to the exact situation and rules that applied in it (provided by the record-and-replay function) is the most important aspect of this study.

Bibliography

[1] T. Walker, *A Behaviour Based Framework for the Control of Autonomous Mobile Robots*, Perth, Australia: Centre for Intelligent Information Processing Systems, UWA, 2006.

[2] R. C. Arkin, *Behaviour-Based Robotics*, Cambridge, MA, USA: The MIT Press, 1998.

[3] C. Watkins, P. Dayan, "Q-learning, technical note," *Machine Learning*, Boston, MA, USA: Kluwer Academic Publishers, 1992, vol. 8, pp. 279–292.

[4] L. J. Lin, "Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching," *Machine Learning*, Boston, MA, USA: Kluwer Academic Publishers, 1992, vol. 8, pp. 69–97.

[5] I. Horswill, "Cerebus: A Higher-Order Behavior-Based System," *AI Magazine*, 2001. [Available from `http://www.cs.northwestern.edu/~ian/cerebus-ai-mag.pdf`]

[6] I. Horswill, R. Zubek, A. Khoo, C. Dac Le, S. Nicholson, "The Cerebus Project," 1998. [Available from `http://citeseer.ist.psu.edu/481460.html`]

[7] H. Van Brussel, R. Moreas, A. Zaatri, M. Nuttin, "A Behaviour-Based Blackboard Architecture for Mobile Robots," in *Proceedings of the 24th Annual Conference of the IEEE*, Aachen, Germany, September 1998, vol. 4, pp. 2162–2167. [Available from IEEE Xplore]

[8] R. Beale, T. Jackson, *Neural Computing: An Introduction*, London, UK: CRC Press, 1990.

[9] R. Sutton, A. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA, USA: The MIT Press, 1998. [Available from http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html]

[10] U. Schnepf, A. Asteroth, M. S. Fischer and K. Möller, "Tracking and Grasping of Moving Objects: a Behaviour-Based Approach," in *Proceedings of the 5th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-92)*, Paderborn, Germany, June 1992, pp. 195–204.

[11] C. Malcolm, "The SOMASS System: a Hybrid Symbolic and Behaviour-based System to Plan and Execute Assemblies by Robot," in *Proceedings of the AISB Conference*, Sheffield, UK, April 1995.

[12] J. H. Connell, "SSS: a hybrid architecture applied to robot navigation," in *Proceedings, IEEE International Conference on Robotics and Automation*, Nice, France, 12–14 May 1992, vol. 3, pp. 2719–2724. [Available from IEEE Xplore]

[13] P. Dario, R. Dillman, H. Christensen, et al, *EURON Research Roadmaps*, Heverlee, Belgium: EURON, 2004. [Available from http://www.cas.kth.se/euron/euron-deliverables/ka1-3-Roadmap.pdf]

[14] US Department of Energy, *Robotics and Intelligent Machines in the U.S. Department of Energy—A Critical Technology Roadmap*, Livermore, CA, USA: Sandia National Laboratories, 1998. [Available from http://www.sandia.gov/isrc/RIMfinal.pdf]

[15] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, March 1986, vol. 2, no. 1, pp. 14-23. [Available from IEEE Xplore]

[16] R. A. Brooks, “Elephants Don’t Play Chess,” *Robotics and Autonomous Systems*, vol. 6, pp. 3–15, 1990. [Available from http://people.csail.mit.edu/brooks/papers/elephants.pdf]

[17] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, M. G. Slack, “Experiences with an Architecture for Intelligent, Reactive Agents,”, *Journal of Experimental & Theoretical Artificial Intelligence*, April 1997, vol. 9, nos. 2 & 3, pp. 237–256. [Available from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.9822&rep=rep1&type=pdf]

[18] W. D. Smart, L. P. Kaelbling, “Effective Reinforcement Learning for Mobile Robots,” in *International Conference on Robotics and Automation*, Washington, DC, USA, May 11–15, 2002, vol. 4, pp. 3404–3410. [Available from http://people.csail.mit.edu/lpk/papers/2002/SmartKaelbling-ICRA2002.pdf]

[19] L. P. Kaelbling, M. L. Littman, A. W. Moore, “Reinforcement Learning: A Survey,” *Journal of Artificial Intelligence Research*, 1996, vol. 4, pp. 237-285. [Available from http://people.csail.mit.edu/lpk/papers/rl-survey.ps]

[20] R. A. Brooks, *Robotics in Space Exploration*, (video lecture), MIT Museum, Cambridge, MA, USA, 10 January 2006. [Available from http://mitworld.mit.edu/video/337/]

[21] R. A. Brooks, the Talking Robots team, “Rodney Brooks - The Past and Future of Behavior Based Robotics,” (audio interview) in *Talking Robots*, 27 April 2007. [Available from http://lis.epfl.ch/resources/podcast/2007/04/rodney-brooks-past-and-future-of.html]

[22] L. P. Kaelbling, *Why Robbie Can’t Learn: The Difficulty of Learning in Autonomous Agents*, (video lecture), EECS Colloquium Series, MIT, Cambridge, MA, USA, 19 November 2001. [Available from http://mitworld.mit.edu/video/45/]

[23] H. L. Dreyfus, S. E. Dreyfus, *Mind over Machine*, New York, NY, USA: The Free Press, 1986.

[24] iRobot Corporation, *Our History*, Bedford, MA, USA: iRobot Corporation, 2008. [Available from http://www.irobot.com/sp.cfm?pageid=203]

[25] S. Zhurenko, "Meet ASIMO the robot," in *Robots that help people*, May 2007. [Available from http://robotsnews.blogspot.com/2007/05/meet-asimo-robot.html]