Defence Research and Development Canada
Recherche et développement pour la défense Canada

Reinforcement learning in mobile robot navigation

Literature review

I. Vincent
DRDC Suffield

Technical Memorandum
DRDC Suffield TM 2006-117
December 2006

Canada

Reinforcement learning in mobile robot navigation

Literature review

I. Vincent
Defence R&D Canada – Suffield

Defence R&D Canada – Suffield

Technical Memorandum

DRDC Suffield TM 2006-117

December 2006

Principal Author

Original signed by I. Vincent

I. Vincent

Approved by

Original signed by D. Hanna

D. Hanna

Head/Autonomous Intelligent Systems Section

Approved for release by

Original signed by P. D'Agostino

P. D'Agostino

Head/Document Review Panel

Abstract

Robotics is gaining popularity in military operations to facilitate soldier tasks and decrease their exposure to dangerous situations. Researchers are currently working on autonomous and semi-autonomous robots to provide soldiers with more intelligent robotic vehicles. At DRDC Suffield, the Autonomous Intelligent Systems Section is building an expertise in intelligent mobility for unmanned ground vehicles by developing robotic platforms that autonomously generate useful locomotive behaviours. One objective of the group is to solve the problem of navigability of shape-shifting mobile robot platforms. Learning to choose the appropriate geometric configuration with respect to the environment is a promising solution. This document presents a literature review emphasizing reinforcement learning in mobile robot navigation. A variety of algorithms are examined in detail such as Q-learning, HEDGER, SRV, RL-LSTM, CQ-L, HQ-L and W-learning. Finally, the applicability of those algorithms to the problem of shape-shifting mobile robot navigation is discussed.

Résumé

La robotique gagne en popularité dans le domaine des opérations militaires parce qu'elle facilite la tâche du soldat et diminue l'exposition de ce dernier aux situations dangereuses. Les chercheurs travaillent actuellement sur les robots autonomes et semi-autonomes visant à fournir aux soldats des véhicules robotiques plus intelligents. La Section des systèmes autonomes intelligents à RDDC développe une expertise en mobilité intelligente des véhicules terrestres sans pilote, en mettant au point des plates-formes robotisées qui génèrent, de manière autonome, des comportements locomoteurs utiles. Un des objectifs du groupe est de résoudre le problème de navigabilité des plates-formes de robots mobiles à configuration variable. Apprendre à sélectionner la configuration géométrique appropriée à l'environnement est une solution prometteuse. Ce document présente une étude de la documentation axée sur l'apprentissage par renforcement en matière de navigation de robots mobiles. Une variété d'algorithmes a été examinée en détail dont Q-Learning, HEDGER, SRV, RL-LSTM, CQ-L, HQ-L et W-Learning. Enfin, on y discute de l'applicabilité de ces algorithmes au problème de la navigation de robots mobiles à configuration variable.

This page intentionally left blank.

Executive summary

Reinforcement learning in mobile robot navigation

I. Vincent; DRDC Suffield TM 2006-117; Defence R&D Canada – Suffield; December 2006.

Background: The Autonomous Intelligent Systems Section is building an expertise in intelligent mobility for unmanned ground vehicles to provide the Canadian Forces with robotic platforms that autonomously generate useful locomotive behaviours. One objective of the group is to solve the problem of navigability of shape-shifting mobile robot platforms.

Principal results: This document presents a literature review emphasizing learning in mobile robot navigation. As the amount of work in this research area is broad, this document focuses on reinforcement learning and behaviour selection. A variety of algorithms are examined in detail such as Q-learning, HEDGER, SRV, RL-LSTM, CQ-L, HQ-L and W-learning. Finally, the applicability of those algorithms to the problem of shape-shifting mobile robot navigation is discussed.

Significance of results: This survey provides the defence communities with an overview of reinforcement learning in unmanned ground vehicles. Researchers interested in developing machine learning algorithms for mobile robot applications may use this review to evaluate the different methods currently applied in robot navigation.

Future work: The reviewed algorithms will be used for reference purposes during the development of a learning architecture to select the optimal shape-shifting robot configurations to negotiate obstacles and generate improved locomotion patterns.

Sommaire

Reinforcement learning in mobile robot navigation

I. Vincent; DRDC Suffield TM 2006-117; R & D pour la défense Canada – Suffield; décembre 2006.

Contexte: La Section des systèmes autonomes intelligents développe une expertise en matière de mobilité intelligente pour les véhicules terrestres sans pilote, visant à fournir aux Forces canadiennes des plates-formes robotisées qui génèrent des comportements locomoteurs utiles. Un des objectifs du groupe est de résoudre le problème de navigabilité des plates-formes de robots mobiles à configuration variable.

Résultats principaux: Ce document présente une revue de la littérature axée sur l'apprentissage machine dans la navigation de robots mobiles. La quantité de travail au profit de cette recherche étant abondante, ce document se concentre sur l'apprentissage par renforcement et la sélection des comportements. Une variété d'algorithmes est examinée en détail dont Q-Learning, HEDGER, SRV, RL-LSTM, CQ-L, HQ-L et W-Learning. On y discute enfin l'applicabilité de ces algorithmes au problème de la navigation de robots mobiles à configuration variable.

Portée des résultats: Cette revue de la littérature fournit à la collectivité de la défense, une vue générale de l'apprentissage par renforcement dans le domaine des véhicules terrestres sans pilote. Les chercheurs intéressés à mettre au point des algorithmes d'apprentissage machine pour des applications de robots mobiles peuvent utiliser cette étude pour évaluer les diffrentes méthodes qui sont actuellement appliquées à la navigation de robots.

Travaux futurs: Les algorithmes étudiés seront utilisés à des fins de référence durant la mise au point d'une architecture d'apprentissage qui servira à sélectionner les configurations optimales de robots à configuration variable, ceci dans le but de négocier les obstacles et de générer des modèles améliorés de locomotion.

Table of contents

List of figures

This page intentionally left blank.

1 Introduction

Robotics is gaining popularity in military operations to facilitate soldier tasks and decrease their exposure to dangerous situations. Currently, robots are used by militaries mainly for research and development purpose. Researchers around the world are working on autonomous and semi-autonomous robots to provide soldiers with more intelligent prototypes. The main problems faced are partial comprehension of the world by perception sensors, navigation in uncertain environments, control of actuators to provide useful locomotion in complex terrains, motion and path planning according to robot abilities, and state of the world.

At DRDC Suffield, the Autonomous Intelligent Systems Section is building an expertise in intelligent mobility for unmanned ground vehicles to provide the Canadian Forces with robotic platforms that autonomously generate useful locomotive behaviours. One objective of the group is to solve the problem of navigability of shape-shifting mobile robot platforms. Learning to choose the appropriate geometric configuration with respect to the environment is studied. It assumes a good comprehension of the immediate world, real-time adaptability to new experiences and good coordination of movements.

This document presents a literature review emphasizing learning in mobile robot navigation. As the body of research work in mobile robot learning is broad, this document focuses on reinforcement learning and behaviour selection. The first section presents the objectives of the research project and current research going on about the variable geometry robot paradigm. The second section explains the robot tasks and the environment complexity, and gives some definitions important to understand the following sections. Also, it highlights why reinforcement learning is an attractive methodology for mobile robot applications. Section three presents and discusses different reinforcement learning algorithms that have been applied to mobile robot navigation. Finally, a synthesis highlights the strengths of each algorithm presented for the shape-shifting robot navigation problem.

2 Context

This section presents the objectives of the research project and current research going on in the area of the variable geometry robot paradigm.

2.1 Objectives

The research project aims to solve the problem of navigability of a shape-shifting mobile robot platform. Learning to choose the appropriate geometric configuration with respect to the environment is a promising solution. It assumes a good comprehension of the immediate world, real-time adaptability to new experiences and a good coordination of the behaviours.

2.2 Existing research platforms

The shape-shifting robot paradigm is currently an active research area aiming to increase the capabilities of robots. Engineers at NASA's Goddard Space Flight Center in Maryland have developed a shape-shifting robotic pyramid called TETWalker [1]. The robot alters its shape to flow over rocks or create useful structures such as communication antenna. A research project at the Maersk Institute in Denmark demonstrates various metamorphoses of a prototype robot called ATRON [2]. It reconfigures its individual spherical modules to change its mode of locomotion (walking, rolling or slithering). Another interesting self-reconfigurable platform is the MTRAN developed by the Distributed System Design Research Group at the National Institute of Advanced Industrial Science and Technology in Japan [3]. This modular robot consists of a distributed control mechanism using a central Pattern Generator controller which enables adaptive locomotion.

DRDC Suffield scientists have initiated work in variable geometry paradigm on a platform called the Shape-shifting Tracked Robotic Vehicle (STRV). Figure 1 shows the STRV in different configurations. Although a number of shape-shifting platforms are now being developed, the algorithms to create, coordinate and learn useful shape-shifting behaviours are lacking. Therefore, the purpose of this project is to investigate the possibility of applying algorithms used in other kind of applications to a shape-shifting mobile robot.

3 Problem definition

The robot relies on its sensors to make decisions and navigate within the world. Its mobility is influenced by the terrain geometry and physical properties. The terrain types vary from well-structured to complex unstructured. Imagine an autonomous vehicle navigating into an office building; then imagine driving into the same building collapsed or after vandalism. Now, imagine the robot going up a circular staircase, down a pipe or into a trench, all scenarios showing the complexity of environments a military robot may have to deal with. As the environment is changing over time and many types of environment are possible, it is unrealistic to attempt to model it. The robot has to adapt and make decisions in real-time. The control and motion planning of robotic platforms in rough terrain with varying terrain conditions, sensor measurement uncertainty and error, and limited computation resources is very challenging [4]. A robot must contend with difficulties such as sensory errors and uncertainties, planned paths not accurately followed, and difficulty distinguishing traversable from untraversable regions since it depends on the vehicle mobility characteristics.

Every embedded system perceives the world through sensors. A typical robotic platform relies on different sensors such as a laser rangefinder for 3-D terrain mapping, an inertial navigational unit for angle measurements, gyro rates and accelerations, a DGPS for localization in outdoor applications, etc. The collection of input signals from the sensors describe the state of the environment. The system may have effectors to influence that state. Those are usually motors and actuators. The embedded system continuously maps the sensory inputs to the effectors outputs. An action is an effector output applied to the environment by the system. Since the sensory inputs are characterized by uncertainty, error, inconsistency

Figure 1: *Pictures of the STRV in different configurations.*

and noise, the robot's understanding of the state of the world is partial. This means that taking the same action in the same perceived state may have a different outcome. This is called a nondeterministic environment. In contrast, a deterministic environment generates the same result for a similar state-action pair.

A self-reconfigurable robot must control its effectors to generate useful locomotion patterns. By sequencing different shapes it can progressively adapt to the terrain geometry and negotiate obstacles. In this way, the variable geometry robot has the ability to traverse obstacles that are untraversable for a "single-configuration" robot. Modeling the robot and the complex environment in an attempt to proceed directly to programming of shape shifting for adaptation to environment variations is unrealistic since there are an infinite number of possible states, as well as a large number of possible actions. Preparation of a lookup table with all state-action pairs is also unrealistic. This problem requires a learning process that can deal with continuous state and action spaces.

Why learning? As explained by Kaelbling [5]:

> *"The problem of programming an agent to behave correctly in a world is to choose some behaviour, given that the rest of the parameters of the agent and world are fixed. If the programmer does not know everything about the world, or if he wishes the agent he is designing to be able to operate in a variety of different worlds, he must program an agent that will learn to behave correctly."*

For this reason, roboticists have focused their efforts on learning algorithms to control robot behaviours. Research in machine learning has seen an increase of interest in the past 25 years. This is partly due to computational resource improvements. Kaelbling [5] explores machine learning algorithms in the context of designing embedded systems that adapt their behaviour to a changing environment.

The previous paragraph introduces the notion of agent. The definition of an agent used in this survey is from Jacques Ferber's general definition of agent [6]:

> *"An agent is a physical or virtual entity*
>
> 1. *which is capable of acting in an environment;*
> 2. *which can communicate directly with other agents;*
> 3. *which is driven by a set of tendencies (in the form of individual objectives or of a satisfaction/survival function which it tries to optimize);*
> 4. *which possesses resources of its own;*
> 5. *which is capable of perceiving its environment (but to a limited extent);*
> 6. *which has only a partial representation of its environment (and perhaps none at all);*
> 7. *which possesses skills and can offer services;*
> 8. *which may be able to reproduce itself;*

9. *whose behaviour tends towards satisfying its objectives, taking account of the resources and skills available to it and depending on its perception, its representation and the communications it receives."*

The agent navigates through the world to reach some goals by learning to map its sensory inputs to its effectors outputs. What goals does the robot need to achieve? For mobile robot navigation, the main goal or objective is, given an initial location, to reach a specific destination. Other goals are related to this primary goal. For instance, the robot can endeavour to avoid obstacles or may opt to cross the obstacle using its locomotion capabilities. Also, it may optimize the path length or the energy consumption. This brings the notion of heterogeneous goals. An agent may have to deal with conflicting goals in which case it has to prioritize one goal over the others.

Finally, the problem definition also includes robot design considerations. A major difficulty arising from the STRV design is that the front tracks may obstruct the field of view of the camera. In particular when position of the front tracks is higher than the body. Therefore, the sensed state is incomplete and the controller may not understand that this state is the same as a previous one, and won't execute the most appropriate action. What happens if the tracks are oriented so the body is tilted? The camera may paint toward the sky instead of looking in the direction of motion. How an agent can determine the optimal action to execute if it doesn't have the appropriate current state of the world? This research project does not aim to solve perception problems. However, it will be interesting to determine the robustness of a learning system given these perception problems.

Since mobile robots navigate in a complex, inconsistent and nondeterministic environment, learning the world's state-action transition function is quite complex. Furthermore, the robot needs to learn and adapt since its sensors provide potentially incomplete, disparate and time-varying representations of the environment. Finally, common machine learning algorithms expect training examples to be of the form (state,action). However, for a real-time mobile robot autonomously exploring the world, training examples are usually not available under this form. For these reasons, roboticists have been focusing their efforts in a particular branch of machine learning called reinforcement learning. This methodology learns an action-map that maximizes reinforcement with training examples of the form *r(s,a)*, where the reinforcement r is function of the current state s and action a. Smart [7] defines a reinforcement signal as: "a mapping from each state of the environment into a scalar value encoding how desirable that state is for the agent." The subsequent sections review reinforcement learning algorithms that have been employed for mobile robot navigation.

4 Reinforcement learning

Reinforcement learning can be defined as follows:

An agent perceives the world, with noise and uncertainty associated with its sensors. Through trial-and-error interactions with the environment, the learning system maps the inputs to some rewards or penalties, and learns the desirability

of being in various states. Then, the controller chooses the optimal action to execute based on the information learned.

An autonomous agent has sensors to observe the world and a set of actions that it can perform to alter its state and achieve some goals. Reinforcement learning addresses the problem of determining the optimal actions to execute in order to achieve its goals. Each time the agent performs an action, a reward indicates the desirability of the resulting state.

In her survey of reinforcement learning, Kaelbling [8] describes two main strategies to solve reinforcement learning problems. One is to search among the behaviours for the one that performs well in the environment. A behaviour is a mapping from the sensory inputs to an action or a sequence of actions. The other strategy is to use statistical techniques and dynamic programming methods to estimate the utility (the value or the cost) of taking a specific action in a particular state of the environment.

4.1 Q-learning

Q-learning is an algorithm that can learn the optimal control strategies from the obtained rewards. The best control policy is the one that selects the actions that maximize the agent's cumulative reward. Q-learning is applicable to systems with deterministic and nondeterministic outcomes of their actions, and to problems with no prior knowledge of the domain theory.

The control policy is the target function. It outputs the optimal action given the current state. A trainer assigns immediate or delayed rewards expressing the desirability of the actions. According to the actions it chooses to execute, the agent can favour exploration of unvisited states and actions to get new information, or exploitation of high rewards states and actions already learned to maximize the cumulative reward. Finally, Q-learning is often used for applications involving partially observable state.

4.1.1 Markov decision process

An agent interacts with its environment. This interaction is represented in Figure 2. At each time step, an agent perceives a state s and chooses to perform an action a. The environment provides the agent with a reward $r = r(s, a)$ reflecting the desirability of the chosen action in the current state, and produces the next state $s' = \delta(s, a)$. Functions r and δ are generated by the environment and are not necessarily known to the agent. In a Markov decision process (MDP), the two functions depend only on the current state and actions, and not on previous states or actions. A MDP is characterized as follows:

- The environment is defined as a finite set of discrete states.
- The environment evolves probabilistically.
- For each state, there is a finite set of actions that the agent may choose.
- For each action taken, a reward is obtained.

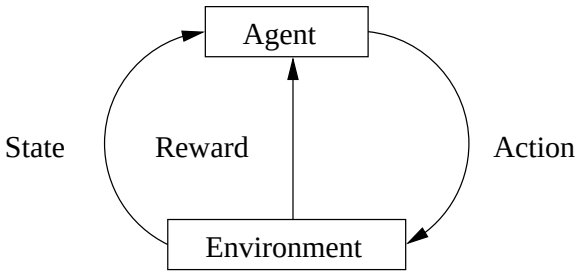

Figure 2: *Representation of the agent-environment interaction.*

- The agent perceives states, takes actions and receives rewards at discrete times.

Globally, the Markov property dictates that the transition probability from one state to another depends entirely on the current state and action. It means that the current state provides all the information required by the agent to select an appropriate action.

Several learning algorithms apply to MDPs. However, real world mobile robots usually deal with continuous state and action spaces rather than a finite set of discrete states. In a complex world, achievement of complex tasks with the MDP assumption is unrealistic. Reinforcement learning in continuous state and action spaces is a current research challenge.

4.1.2 Learning the optimal control strategy

To interact with the environment and reach its goals, the agent must learn a policy π that maps states into actions, $\pi : S \rightarrow A$. If the agent follows a policy from an initial state s_t, the discounted cumulative reward V^{π} achieved by the policy from that state is expressed by:

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... \tag{1}$$

where the discount factor γ ranges from 0 to 1. This factor reflects the relative importance of immediate versus delayed rewards. The agent should learn the policy maximizing this cumulative reward over time for all states. The resulting optimal policy π^* is as follow:

$$\pi^* = \arg\max_{\pi} V^{\pi}(s), \forall s \tag{2}$$

This formula of the optimal policy is difficult to apply directly, since the training data are usually not available of the form state-action pairs. Instead, the learning system receives a sequence of rewards and must deduce from it the best action to choose. The Q function addresses this problem.

4.1.3 Q function

Starting from state s and applying action a, the maximum discounted cumulative reward that can be achieved is the evaluation function value $Q(s, a)$. It is defined as:

$$Q(s, a) \equiv r(s, a) + \gamma V^{\pi^*}(\delta(s, a)) \tag{3}$$

The optimal policy is then the maximum Q value over the entire set of possible actions for that state,

$$\pi^*(s) \equiv \arg\max_a Q(s, a). \tag{4}$$

This formula provides the agent with the possibility of learning the Q function instead of the V^{π^*} function and thus determining the optimal action without prior knowledge of the r and δ functions. It only needs to choose the action maximizing $Q(s, a)$.

Figure 3: Algorithm for deterministic MDP (from reference [9])

for *each state-action pair* **do**
  Initialize the table entry $\hat{Q}(s,a)$ to zero.
end
Observe the current state s.
repeat
  Select an action a and execute it.
  Receive an immediate reward r.
  Observe the subsequent state s'.
  Update $\hat{Q}(s,a)$ using the following training rule, where a' are the possible actions in s':

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s, a') \tag{5}$$

$$s \leftarrow s' \tag{6}$$

until *end of run*

Figure 3 presents the Q-learning algorithm for a deterministic MDP. It iteratively approximates the Q values. A table stores those values. There is an entry for each state-action pair. $\hat{Q}(s, a)$ converges to Q if the system is a deterministic Markov decision process and it converges asymptotically with state-action pair visits. Moreover, the reward function must be bounded.

4.1.4 Nondeterministic systems

In the presence of noisy sensors and effectors, it is more appropriate to model actions and rewards as nondeterministic. In this case, $r(s, a)$ and $\delta(s, a)$ may have probabilistic

outcomes. The probabilistic distribution depends exclusively on the current state and action. For this reason, the system is called a nondeterministic MDP.

A nondeterministic reward function may generate a different reward each time the same state-action pair is repeated. The value is continuously altered and does not converge to Q. Therefore, a new training rule is required to achieve convergence. It is done by decaying the current $\hat{Q}$ value and the revised estimate. At the nth iteration, $\hat{Q}_n(s,a)$ is given by

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n(r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')). \tag{7}$$

α_n is the learning rate at the nth iteration. It is commonly decayed to give more importance to the first stages of learning. In other words, α_n is inversely proportional to the number of times N that the state-action pair has been visited,

$$\alpha_n = \frac{1}{1+N}. \tag{8}$$

If α_n equals 1, the training rule is the same as for the deterministic case. Although the update of $\hat{Q}$ is more gradual for a nondeterministic system due to a smaller learning rate, it is this reduction of α_n that ensures convergence.

Q-learning represents each state-action pair has an entry in a lookup table. It does not generalize Q values for unseen pairs. Since the convergence is proven only if every state-action pair is visited infinitely often, it is unrealistic for large and continuous spaces. A solution is to combine function approximation methods with Q-learning. For instance, the lookup table may be replaced by a backpropagation neural network using each $\hat{Q}(s,a)$ update as a training example. The neural network is trained to output the target values of $\hat{Q}$ given by eq. 7 as shown below,

$$input\ (s,a)\ \rightarrow\ backpropagation\ neural\ network\ \rightarrow\ output\ \hat{Q}. \tag{9}$$

4.2 Existing algorithms

The literature contains a number of diversified algorithms developed to navigate mobile robots executing tasks such as corridor following, obstacle avoidance, A-to-B mobility using reinforcement learning. Those with the potential of making a contribution to a learning algorithm for the STRV are presented here.

4.2.1 Acceleration of Q-learning for continuous space control tasks

Smart [7] and [10] introduces a Q-learning algorithm for reinforcement learning on mobile robots named the HEDGER prediction algorithm. In continuous state control tasks, it replaces Q values lookup tables by an approximation of the target function. Assuming that

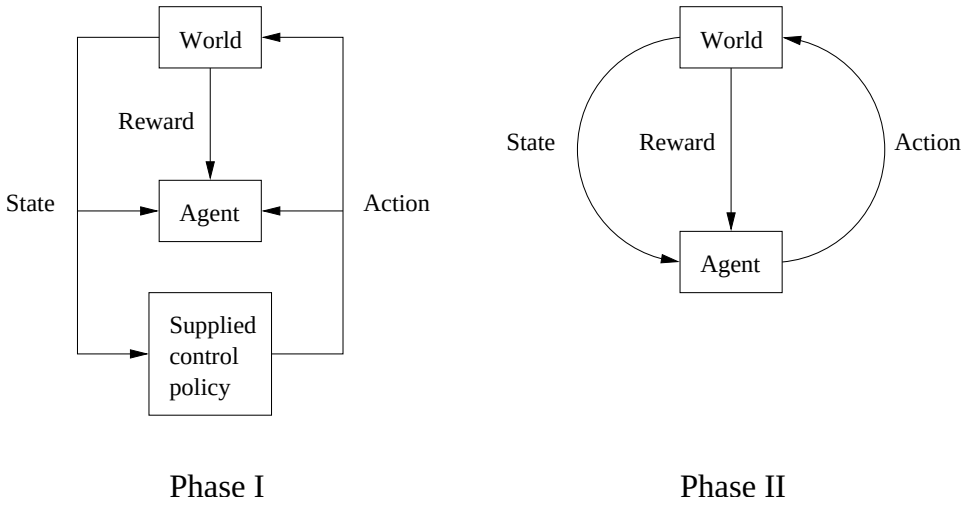

Figure 4: *Learning phases.*

each training example can be represented by a point in an n-dimensional Euclidean space, HEDGER uses locally weighted regression (LWR) for interpolation within the training data, and locally weighted averaging (LWA) if the query point is outside the training data, to predict an approximation of the target function. LWR uses the training examples near the query point to construct a local approximation of the target function in the neighbourhood of that point. The contribution of each training example is weighted according to a function of its distance from the query point. In the HEDGER algorithm, the function is a kernel function with an associated bandwidth. A large bandwidth means that points away have more influence. The result is a smoother approximated function. In contrast, LWA predicts the target function value using all training examples, not just the surrounding points, also weighted by their distance to the query point. The approximation value obtained from LWR or LWA stays the same if no reward occurs. When reinforcement is observed, a standard Q-learning algorithm is used to iteratively improve the approximation value using eq. 7. The HEDGER algorithm learns quickly from a small amount of data. Its effectiveness has been demonstrated on a real robot executing an obstacle avoidance task and a corridor-following task.

Furthermore, the authors split learning into two phases for an application with a sparse reward function and no prior knowledge of the environment. Learning stages are represented on Figure 4. The first phase is a passive learning process where the robot is controlled by a supplied control policy. For instance, a human can drive the robot using a joystick. The learner passively watches the states, actions and rewards. It uses the rewards to bootstrap information into its target function approximation. The second phase of learning starts when the target function approximation can effectively control the robot. In this phase, the learned policy is in control of the robot and learning progresses using a standard Q-learning algorithm. The knowledge acquired in the first stage allows the agent to learn more effectively and reduce the time spent acting randomly in the second phase.

4.2.2 Careful design of the reinforcement functions

As mentioned earlier, mobile robots usually navigate in nondeterministic environments. Therefore, taking the same action in the same state may lead to a different next state and reinforcement. Also, several applications are characterized by multiple goals, noisy states and inconsistent reinforcement. The rewards incurred may be immediate, intermittent and/or delayed. For these reasons, it is challenging to learn a world model and control a robotic platform navigating that kind of world. Mataric [11] discusses the reasons why traditional reinforcement learning methods, temporal differencing applied to deterministic MDP, perform poorly in such domains. The author highlights how it is important to carefully design the reinforcement functions if the robot is to be able to learn successfully. She proposes a methodology that embeds domain knowledge into the reinforcement using heterogeneous reinforcement functions, reinforcing each goal reached individually, and progress estimators, providing a measure of improvement relative to a specific goal. The method takes advantage of implicit domain knowledge to accelerate learning by converting reward functions into error signals. A matrix $A(c, b)$ stores the value of the appropriateness of a behaviour b associated with a situation c. The values change over time t according to the total reinforcement R obtained.

$$A(c, b) = \sum_{t=1}^{T} R(c, t) \tag{10}$$

Each progress estimator I evaluates the progress done accomplishing a specific goal. If n progress estimators are designed, and E is the sum of the heterogeneous immediate rewards, the total reinforcement R gained by the agent at time t is given by:

$$R(c, t) = \sum_{i=1}^{n} w_i I_i(c, t) + w_{n+1} E(c, t) \tag{11}$$

Rewards are weighted according to the accomplished progress. The weights w_i correspond to the contribution of each component to the overall reinforcement. The approach is experimentally compared to traditional reinforcement learning methods in a mobile robot group learning a foraging task. It has been demonstrated that even in the presence of noise, multiple reinforcers and progress estimators significantly accelerate learning in single and multiple agent applications.

4.2.3 Generating real-valued actions

Gullapalli [12] discusses how to increase the applicability and efficiency of direct, or model-free, reinforcement learning methods. As shown in Figure 5, a direct method perturbs the controller's actions and observes the consequences on the performance metric, reinforcement, to gain training information. The author has developed a neural network reinforcement learning algorithm applicable to associative tasks with continuous action space. It is

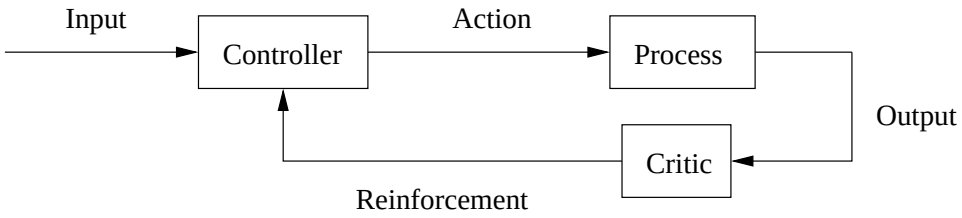

Figure 5: *Direct reinforcement learning diagram. By perturbing the action, the evaluation (reinforcement) changes.*

called the stochastic real-valued unit (SRV). Depending on the input, the agent selects an action which produces a reward. This reward is then used to update the action probability. The controller outputs the action with the highest expected reinforcement. The SRV algorithm does not use Q-learning. Instead, the controller computes action y_n as a random variable depending on a Gaussian distribution with mean μ_n and standard deviation σ_n.

$$y_n \sim N(\mu_n, \sigma_n) \tag{12}$$

The algorithm adjusts the mean and the standard deviation to increase the probability of selecting the optimal real-valued action for each input i. Two internal parameter vectors, θ_n and ϕ_n, and the inputs x_n are used to compute μ_n and σ_n.

$$\mu_n = \sum_{i=1}^{k} \theta_{ni} x_{ni} \tag{13}$$

$$\hat{r}_n = \sum_{i=1}^{k} \phi_{ni} x_{ni} \tag{14}$$

$$\sigma_n = s(\hat{r}_n) \tag{15}$$

The function s is a monotonically decreasing nonnegative function and $\hat{r}_n$ is the expected reward. When the maximum reinforcement is expected, $\hat{r}_n = 1$, the standard deviation is zero, $s(1) = 0$. The mean and the variance are adjusted according to previous experiences. This is possible by updating the values of θ_n and ϕ_n according to the reward $r(y_n, x_n)$ received from the environment, and the learning rate α, as shown below,

$$\theta_{n+1} = \theta_n + \sigma_n(r(y_n, x_n)\hat{r}_n)(y_n - \mu_n)x_n, \tag{16}$$

$$\phi_{n+1} = \phi_n + \alpha(r(y_n, x_n)\hat{r}_n). \tag{17}$$

A high σ_n happens when the action performs poorly and the system opts for exploring the other possible actions. When the action performs well, σ_n decreases and the mean moves towards the action. The result is a tendency to generate more action values near the successful one. This algorithm can learn associative maps from input vectors to real-valued actions without the necessity of having the desired responses available to the algorithm. The SRV unit has been successfully applied to learn to balance a rigid pole mounted on a mobile cart by applying forces to the cart. This involves learning to control an unstable dynamic system. This algorithm has been presented here, even though it has not been applied to mobile robot navigation, since the concept is applicable to the control of unstable systems, like the STRV, and it can map continuous action space.

4.2.4 Fuzzy rules tuned by reinforcement learning

For some applications, it is easier to represent the human approach using linguistic rules. Fuzzy logic can be used to translate the human approach into fuzzy sets to control robotic systems. Zavlangas [13] and Saffiotti [14] present mobile robot applications in which collision-free trajectories are executed using fuzzy logic. They highlight that it is hard to derive and tune the rules and membership functions. They have investigated the possibility of using the learning capability of neural networks to reduce the amount of work spent on deriving and tuning fuzzy logic controllers. Several algorithms have been developed using a combination of fuzzy logic and reinforcement learning to make the algorithm learn the fuzzy rules and membership functions. In general, a switch module decides the optimal action to perform by comparing the behaviours proposed by the different fuzzy agents. The advantages of fuzzy controllers are the simplicity and short time response. For these reasons, they are suitable for real-time applications. In parallel, neural networks can learn so they are well suited to adapt variables to changing conditions.

Yung [15] presents a learning algorithm for mobile robot navigation in complex and cluttered environments. The method combines fuzzy logic and reinforcement learning. The navigator consists of three modules: obstacle avoidance, move-to-goal and a fuzzy behaviour supervisor. The real-time obstacle avoidance module learns the fuzzy rules to avoid obstacles through reinforcement learning. Sensory inputs are fuzzified, then rules are constructed using reinforcement learning, decisions are made, and, finally, defuzzification generates the output actions. The procedure is similar for the move-to-goal module. The third module fuses the two behaviours and generates the appropriate motion commands for the robot. Simulation of a corridor-following task has demonstrated that this method learns the rules more efficiently than the Environment Exploration Method (EEM). EEM operates the robot to explore the world and has the rules constructed without using reinforcement learning. It is time consuming and in most of the cases, an insufficient number of rules are learned. Tuning the fuzzy rules by reinforcement learning accelerates the construction of rules and improves convergence.

4.2.5 Memory in reinforcement learning

In robot applications, the states are usually partially observable (Partially Observable Markov Decision Processes). The robot must not only learn to map states to actions, but it must also memorize past experiences to determine the current state. To deal with this problem and others such as large and continuous environments, and limited numbers of learning experiences, Bakker [16] combines reinforcement learning with memory, in an unsupervised learning system doing event extraction. Also, it uses online exploration to proceed to offline policy learning.

The methodology presented learns to extract discrete events characterized by significant sensory input changes and store them. It uses the Adaptive Resource Allocation Vector Quantization (ARAVQ) network to proceed to event extraction. It is an unsupervised learning method that classifies sensory input vectors into a set of classes to reduce the amount of data stored for world representation. The classes are associated with memorized model vectors. Each model vector represents an event class. ARAVQ averages n sensory input vectors to reduce noise impact. A new model vector is allocated when a new stable average sensor vector is encountered. It is considered stable if the average Euclidian distance between the average vector $\bar{x}(t)$ and the n input vectors, $d_{\bar{x}(t)}$, is less than a certain threshold. Furthermore, the average vector must be novel, i.e the average Euclidian distance between the best matching stored model vector and the n input vectors must be larger than $d_{\bar{x}(t)}$ plus a distance parameter δ. On reception of a stable input $\bar{x}(t)$, it is compared to the j model vectors m_j to select the class to which it belongs. The winning model vector $v(t)$ is selected according to:

$$v(t) = \arg\max_j \{\|\bar{x}(t) - m_j\|\} \tag{18}$$

An event occurs when the winning model vector changes, and then ARAVQ sends information to the reinforcement learning component to learn the policy. Bakker [17] concludes that the ARAVQ network has two advantages. First, it simplifies the high-dimensional sensory inputs and filters out noise at a certain level. Second, it compresses the long, high sampling rate timeseries into a compact sequence of discrete events.

In the reinforcement learning part of reference [17], the methodology presented uses a Long Short-Term Memory recurrent neural network (LSTM) in conjunction with reinforcement learning (RL). The resulting method is called RL-LSTM. Its architecture is presented in Figure 6. Hochreiter and Schmidhuber [18] detail the LSTM method and Bakker [19] applies RL-LSTM to non-Markovian problems. A simplistic explanation of this quite complex algorithm is given here.

LSTM learns what information to memorize and the appropriate times to use it. A memory cell consist of Constant Error Carrousels (CECs), having linear activation functions which do not decay over time, and gates. The schema of a memory cell is presented on Figure 7. It stores information for long periods of time. The input gate learns to open and close access to the CEC at appropriate moments to protect the memory contents from perturbation by

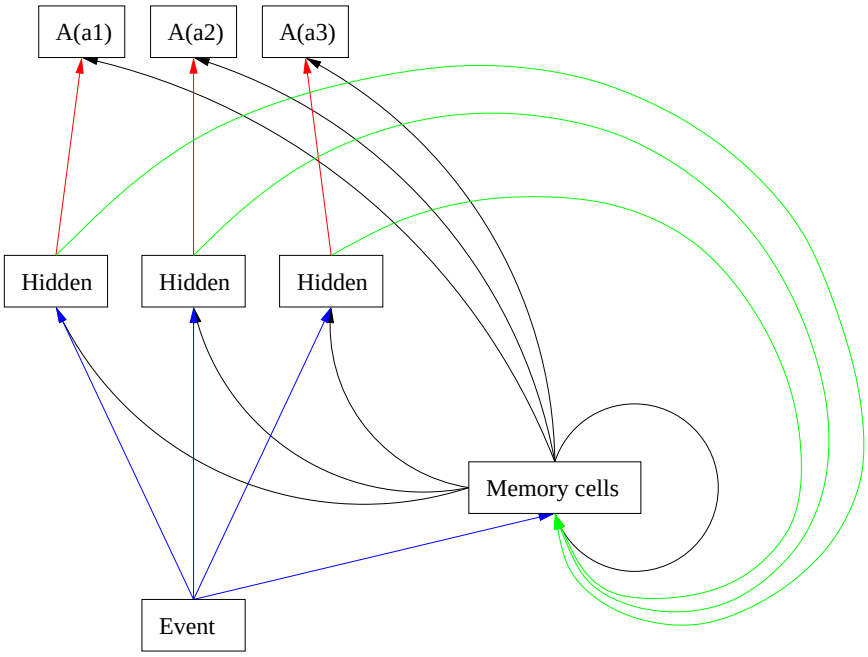

Figure 6: *RL-LSTM Architecture, from reference [16]. Three actions can be selected in this example:* a_1, a_2 *and* a_3.

irrelevant inputs. Similarly, the output gate learns to send the CEC outputs at the right time to protect other units from perturbation by currently irrelevant memory contents. The forget gate learns to reset the CEC activation when the stored information becomes no longer useful.

The LSTM outputs values representing the advantage of performing each action. Instead of providing Q values like in the Q-learning algorithm, the LSTM outputs Advantage values using the Advantage learning algorithm [20]. The RL-LSTM recurrent neural network approximates the value function of the Advantage learning. It computes the Advantage value $A(s, a)$ for action a and state s as follows:

$$A(s,a) = V(s) + \frac{r + \gamma V(s+1) - V(s)}{\kappa} \tag{19}$$

$$V(s) = \max_a A(s,a) \tag{20}$$

where $V(s)$ is the maximum Advantage value over all actions, r is the immediate reward, γ a factor discounting future rewards, and κ a constant scaling the difference between optimal and suboptimal actions. If $\kappa = 1$, the equation becomes equivalent to eq. 3 in Q-learning. At each time step, the controller selects the action with the highest estimated Advantage value. The action is performed until a new event occurs.

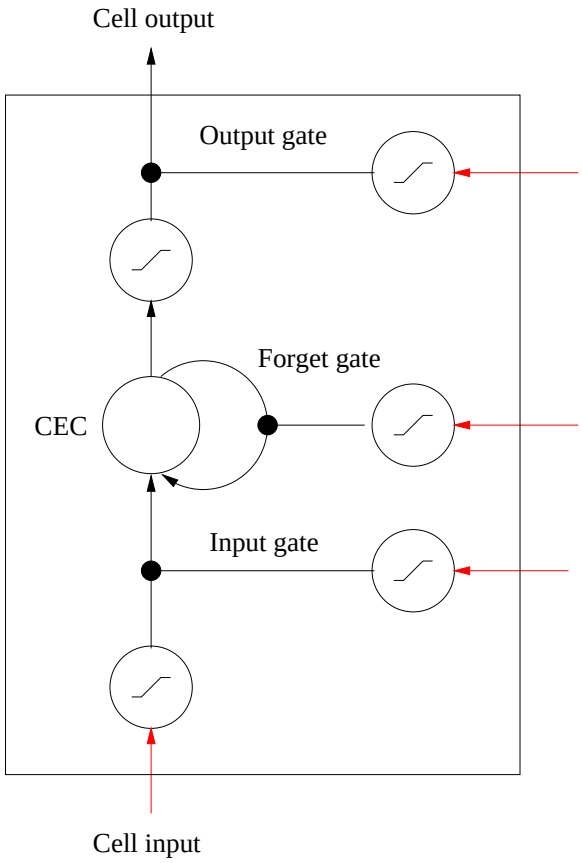

Figure 7: *One memory cell, from reference [16]*

.

To facilitate the learning process, Bakker [16] collects online data from a few real robot explorations and then uses them to train the policy offline. This reduces the amount of real robot runs required to learn a correct policy. Once the network approximation of the optimal values is good enough, then the greedy policy can be followed to achieve optimal behaviours. The algorithm has been tested on a real mobile robot learning to navigate in a T-maze. Each T-junction extremity is a goal position. Depending on a light previously seen in the maze, the robot must reach the left or the right goal position. Rewards are obtained only when the goal positions are reached. The learning task requires the agent to find the temporal dependency between the light and the action to perform at the junction. It also has to learn to store the light information reliably in memory, and to learn to use that information at the appropriate time. ARAVQ extracts sequences of events and the RL-LSTM algorithm learns the regularity between the light information and the best action at the T-junction.

4.3 Behaviour selection

As this project research aims developing an algorithm to learn shape-shifting behaviours for a robotic platform enabling it to negotiate obstacles, it is interesting to look at different algorithms dealing with behaviour selection.

4.3.1 Learning by reinforcement to perform composite tasks composed of sequentially combined subtasks with termination conditions.

Singh [21] and [22] presents a Compositional Q-learning (CQ-L) algorithm. This modular architecture learns by reinforcement learning to perform composite tasks formed by sequencing elemental tasks. Each task is a MDP and has a completion state. The decomposition of the composite task is unknown from the agent. The CQ-L algorithm constructs the Q values of a composite task from the Q values of its elemental tasks. Each elemental task is learned by a module, using Q-learning. A gating module does composite task decomposition by learning to compose elemental task modules over time. A stochastic switch selects one Q-learning module at each time step. CQ-L outputs the sum of the selected Q-learning module output and the output of a bias module. Figure 8 presents a simplified version of the CQ-L architecture diagram presented in reference [21].

Singh [22] tests the CQ-L algorithm in a simulated 2D navigational mobile robot task. The robot has to reach a destination via a sequence of sub-goal locations. Tham [23] extends the concept to a more general reward function also rewarding states different than the goal state, and to an agent with more than one actuator. The architecture is tested on a simulated two-linked manipulator with a large state and action space. The manipulator has to reach specified destinations. Two issues are studied: how to acquire elemental skills to solve a task and how to coordinate these skills to perform complex tasks. The concept of CQ-L needs prior knowledge of useful sub-sequences. Those sub-sequences are learned through experience in learning to solve other tasks.

CQ-L deals with subtasks having termination conditions and combined sequentially. When

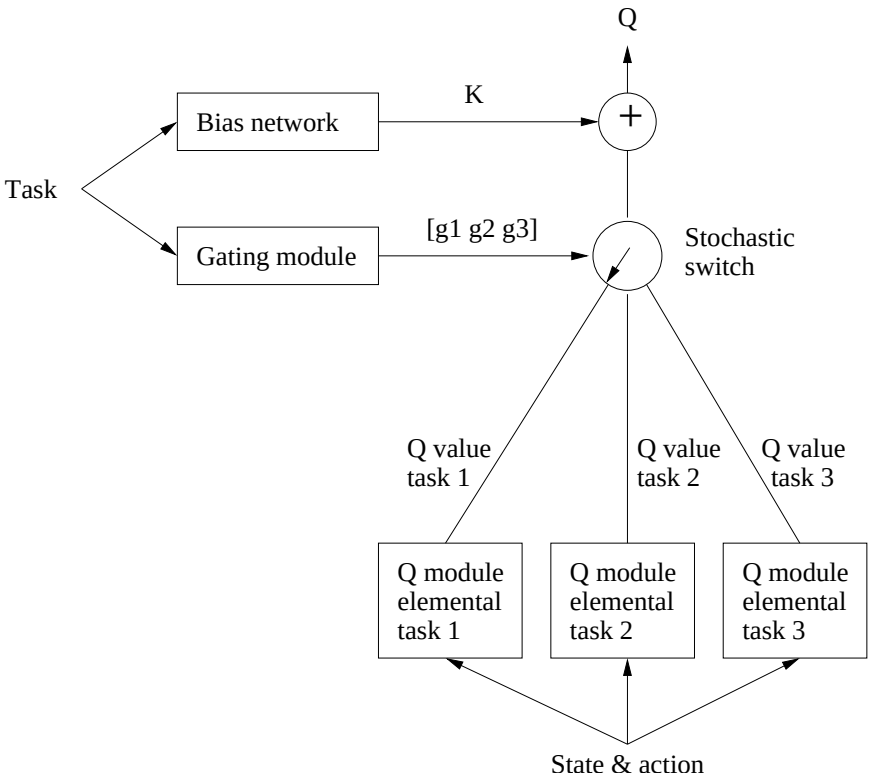

Figure 8: *CQ-L architecture from reference [21]. The Q modules learn the Q values for the elemental tasks (3 in this figure). For each Q module, the gating module outputs the probability of selecting that particular Q module. The bias module learns K. It is function of the composite task and the elemental task position in the decomposition.*

a mobile robot must select actions to execute tasks acting in parallel and interrupting each other before their completion, other approaches are more suitable.

4.3.2 Learning composite tasks having conflicting subtasks acting in parallel

Lin [24] proposes a Hierarchical Q-learning architecture (HQ-L) consisting of a group of Q-learning agents that learn the sub-problems of a main problem. When the robot observes state s, each agent A_i suggests an action a_i. A switch chooses a winner A_k and executes the corresponding action a_k. Furthermore, a decision making Q-learning agent learns the Q values $Q(s, i)$ of the best agent i to select in state s. This Q value is easier to learn than Q(s,a). The agents learn their Q-values simultaneously. The update of the Q-values is as follow for the winning agent:

$$Q_k(s, a_k) := (1 - \alpha)Q_k(s, a_k) + \alpha(r_k + \gamma \max_{b \in A} Q_k(y, b)) \tag{21}$$

and for the others:

$$Q_i(s, a_k) := (1 - \alpha)Q_i(s, a_k) + \alpha(r_i + \gamma \max_{b \in A} Q_i(y, b)) \tag{22}$$

where y is the subsequent state and b is an action included in a set of actions A.

Humphrys [25] compares a standard Q-learning agent to a HQ-L system in a simulation of a house robot application. The HQ-L architecture requires less memory since each agent only senses the subspace that is relevant to its reward function, and builds up Q values quicker than the standard Q-learning algorithm.

4.3.3 Reinforcement learning and action-selection with conflicting heterogeneous goals

Humphrys [25] presents a decentralized action-selection method for applications with conflicting heterogeneous goals. The author introduces a reinforcement learning method called W-learning. The action-selection process is learned while the W-learning modules compete to control the system. The concept is that different agents modify their behaviour based on whether or not they are succeeding in getting the robot to execute their action. The author calls it the "minimize the worst unhappiness" strategy.

This method is based on Brooks' horizontal subsumption architecture [26]. Each agent (layer) can function in the absence of the others. An agent does not know about the global system. When a state s is observed, each agent i suggests an action with its corresponding strengths $W_i(s)$ or W values. The W value indicates how important a specific action is for that agent. A switch selects the highest W value proposed by the agents and executes its corresponding action a_k.

$$W_k(s) = \max_I W_i(s) \tag{23}$$

where I is the set of possible agents. A_k becomes the leading agent in the competition for state s at that moment.

W-learning observes what is happening when an agent is not obeyed and another action is taken. It learns how bad it is to do not be obeyed in this state by observing the reward r and the state s' it led to. The agent i estimates a substitute Q value, $Q_i(s, a_k)$, resulting from having executed action a_k in state s instead of a_i. W-learning is a way of building up this estimate when agents do not share the same suite of actions. When agent A_k is the winner, all other agents update with:

$$W_i(s) \leftarrow Q_i(s, a_i) - (r_i + \gamma \max_{b \in A} Q_i(s', b)) \tag{24}$$

$W_k(s)$ is not updated. For a discrete case of W-learning, each $W_i(s)$ is stored in lookup tables and updated with:

$$W_i(s) := (1 - \alpha)W_i(s) + \alpha(Q_i(s, a_i) - (r_i + \gamma \max_{b \in A} Q_i(s', b))) \tag{25}$$

where A is the set of possible actions. The learning rate α equals successively 1, $\frac{1}{2}$, $\frac{1}{3}$, and so on. The agent learns by experiencing what the others want to do. When the leader changes, the agent is not required to learn a new $Q_i(s, a)$, it just changes the W value.

In this architecture, Q-learning solves the reinforcement learning problem and W-learning solves the action selection problem. The agents begin with random Q values and W values; hence the robot starts by executing random behaviours. Q-learning propagates rewards into Q values, and W-learning propagates the differences between Q values and W values, until the robot executes a steady pattern of behaviours. The leader will not keep changing forever. W-learning attempts to maximize the collective reward.

The W-learning algorithm has been tested in an ant world simulation. It represents the conflict between foraging food while avoiding predators. The algorithm has also been tested on a larger state space in a simulated house robot context.

4.3.4 Behaviour activation by reinforcement and precondition fulfilment

Maes [27] presents an interesting autonomous 6-legged robot that learns to coordinate its legs to walk forward. The behaviour-based algorithm learns by reinforcement learning to activate the different behaviours.

The architecture is decomposed into behaviours. Every behaviour learns when it should be active. This is possible by finding under which conditions the behaviour maximizes

Behaviour B	Active	Inactive
Positive feedback	j	k
No positive feedback	l	m
Negative feedback	a	b
No negative feedback	c	d

Figure 9: *Performance table for an arbitrary behaviour B. It counts the number of times positive and negative feedback does or does not occur when the behaviour is active and inactive.*

positive feedback and minimizes negative feedback, and how relevant it is to the global goal achievement.

Each behaviour keeps track of its performance in tables. Those tables contain the number of times positive and negative feedback happened when the behaviour was active and not active. Figure 9 illustrates the performance table for an arbitrary behaviour.

The correlation between positive feedback and the activation status of a specific behaviour is defined as:

$$Corr(P, A) = \frac{jm - lk}{\sqrt{(m + l)(m + k)(j + k)(j + l)}} \tag{26}$$

where P means positive feedback and A means active status. The same statistics are maintained for negative feedback, therefore $Corr(N, A)$ measures the degree to which the behaviour is correlated with the presence or absence of negative feedback. The relevance of a particular behaviour B is defined as:

$$Relevance(B) = Corr(P, A) - Corr(N, A) \tag{27}$$

It evaluates the probability that the behaviour becomes active. The reliability of a behaviour is defined as:

$$Reliability(B) = \min(\max(\frac{j}{j + l}, \frac{l}{j + l}),\ \max(\frac{a}{a + c}, \frac{c}{a + c})) \tag{28}$$

The closer the reliability is to one, the more consistent the behaviour. The algorithm decides whether the behaviour should improve itself or not based on its reliability.

Other statistics are required to select the appropriate behaviour. Some specific conditions are monitored. Figure 10 illustrates a table for an arbitrary condition monitored. It shows that n is the number of times positive feedback happened when the behaviour was active and the condition was set to ON; similarly for the rest of the table. The next equation evaluates the correlation between a positive feedback P and a condition set to ON.

Active Behaviour B	Condition ON	Condition OFF
Positive feedback	n	o
No positive feedback	p	q
Negative feedback	e	f
No negative feedback	g	h

Figure 10: *Condition table for an arbitrary behaviour B. It counts the number of times positive and negative feedback does or does not occur when the behaviour is active and the condition is ON or OFF.*

$$Corr(P, A, ON) = \frac{nq - po}{\sqrt{(q+p)(q+o)(n+o)(n+p)}} \tag{29}$$

The same equation is applicable to negative feedback and to the OFF condition status. When the system notices a strong correlation between the condition being monitored and a certain feedback, it considers this condition as being a precondition for this particular behaviour. When a new condition is learned, the behaviour becomes active only when this condition is obtained.

The control strategy is to group together the behaviours acting on the same actuator. At every iteration, it is determined if the inactive behaviours have their preconditions fulfilled. Each group can select only one or none of the behaviours based on, in order of importance, their relative relevance, their reliability, and finally, the interestingness of the current situation for the behaviour. The selected behaviours are then activated.

The algorithm was tested on Genghis, a 6-legged robot. A trailing wheel provides the algorithm with positive feedback if the robot moves forward. Front and back touch sensors provide with negative feedback if the body touches the ground. Each leg has a swing-leg-forward and a swing-leg-backward behaviour. An additional behaviour ensures horizontal balance of the platform. All behaviours learn the conditions under which they should become active. The perceptual conditions indicate whether the legs are up or down. The results showed that the system coordinated its members by always keeping three legs on the ground. In addition, the robot learned that although swing-leg-backward doesn't give a negative feedback, it shouldn't become active since it doesn't provide positive feedback. To conclude, the system learned to optimize positive feedback and minimize negative feedback.

5 Synthesis

Mobile robots rely on their sensors to make real-time decisions and navigate in complex terrains. They deal with inconsistent, complex and nondeterministic environments. Therefore, it is difficult for the designers to program robots. The solution is to make the embedded systems learn to behave correctly. Reinforcement learning addresses the problem of how to learn the best action to perform based on rewards and penalties incurred from performing

particular actions.

This literature review presented different reinforcement learning algorithms applied to mobile robot navigation. Most of them derived from the Q-learning algorithm. This method learns the optimal control strategy to select the actions maximizing the agent cumulative reward.

The HEDGER algorithm widens Q-learning capabilities to continuous state space. Instead of lookup tables, it uses function approximations like LWR and LWA to interpolate the Q values from the training examples. Then, standard Q-learning optimizes those Q values. This algorithm could be applied to the STRV for mapping the sensory inputs such as roll, pitch, yaw, acceleration along the x, y and z axe, speed and angular position of the tracks, obstacle height or gap width, ground contact force on the tracks, etc. Those can be represented by a point in an n-dimensional Euclidean space.

The SRV algorithm extends the Q-learning capabilities to continuous action space. It represents actions by Gaussian distributions. When an action performs well, the standard deviation decreases and the average value moves toward this action. The algorithm tends to generate action values near the successful action. This could be applied to control the STRV track angular position for instance. The better a position performs, the narrower is the Gaussian distribution and the more centred it is on this angle value.

Mataric [11] emphasizes the importance of carefully designing reinforcement functions to get useful learning. She proposes using heterogeneous reinforcements and progress estimators to accelerate the learning rate. This means that if the agent gets only sparse rewards, the task may be arduous to learn. Adding intermittent rewards and progress estimations speeds up the process. The STRV research project will involve heterogeneous reinforcement such as goal reached, stable configuration, low energy consumption configuration, etc. It will also need progress estimators like speed of convergence toward the destination location, gain or loss in altitude to step over high obstacles, go down into trenches, follow ascending or descending pipes, etc. All those will accelerate the learning process for controlling the shape-shifting vehicle.

Another way to incorporate the human approach into a learning algorithm is to use fuzzy logic. It integrates the human linguistic to the process. Fuzzy logic controllers have the advantages of being simple and providing short time responses. The main problem is the derivation and tuning of the fuzzy rules. Reinforcement learning addresses this problem. It also adapts the fuzzy controller to changing conditions.

The RL-LSTM algorithm finds dependencies between some events and actions to perform. It combines long short-term memory to reinforcement learning. It uses the ARAVQ algorithm to extract discrete events from sensory inputs. The LSTM learns to store the information reliably in memory and to use it at the appropriate moments. It outputs Advantage values. In contrast with Q-learning, Advantage learning scales the difference between optimal and suboptimal actions. The RL-LSTM recurrent neural network approximates the value function of the Advantage learning. The controller selects the action with the highest esti-

mated Advantage value and the action is performed until a new event occurs. This method remembers past experiences and learns dependencies between past events and actions. The RL-LSTM method is promising for the shape-shifting paradigm. The agent could learn regularities between extracted features from the environment and behaviours to perform. For instance, when the robot sees a step or a staircase, it could learn to relate it to particular stepping behaviours, memorizing the information for further similar situations. It could be interesting to see how well it could find the dependencies between a specific feature seen (stair, pipe, gap, etc.) and the sequence of actions to perform for negotiating the obstacle.

The last part of this literature review covered reinforcement learning in behaviour selection processes. The CQ-L algorithm decomposes composite tasks into elementary tasks. Each task has a completion state and the tasks are performed sequentially. CQ-L has a modular architecture where each elemental task module learns to perform its task using Q-learning. A gating module also uses Q-learning to learn to compose elemental task modules over time to perform composite tasks. The disadvantages of this method are that it needs prior knowledge of useful sub-sequences and the tasks can't act in parallel. Furthermore, it is applicable to Markov decision processes only.

The HQ-L algorithm deals with conflicting subtasks acting in parallel. It has a structure similar to the CQ-L. Q-learning agents learn the sub-problems suggesting an action and a decision-making Q-learning agent learns the best agent to select in a particular state. This algorithm requires less memory than CQ-L for the same problem and can deal with conflicting sub-problems acting in parallel. This algorithm could be used to select the STRV behaviours. Each agent could learn a specific behaviour such as stair up or down, pipe following, snowshoeing, obstacle step-over, gap-crossing, steering, etc. Each agent would suggest an action to perform. The decision-making agent would learn to select the optimal agent in a particular state.

Another action-selection algorithm presented in this document is the W-learning. This method deals with heterogeneous goals and agents. Each agent is a W-learning module and competes to control the system. An agent suggests its action with the highest W value indicating how important this action is for that agent. A switch selects among all the actions proposed by the agents the one with the highest W value. Those values are updated based on what is happening if the agent is not obeyed. In this architecture, Q-learning solves the reinforcement learning problem and W-learning solves the action selection problem. Q-learning propagates rewards into Q values, and W-learning propagates the differences between Q values and W values, until the robot executes a steady pattern of behaviours. This method is interesting for the shape-shifting paradigm. An agent could be taught to perform a specific behaviour like mentioned in the previous paragraph. Then, all agents could compete to control the system and update their W values based on what happens if they are not obeyed. New agents could easily be added to the structure when new behaviours are investigated.

The last algorithm presented is a behaviour-based architecture to control a six-legged walking robot. It learns by reinforcement learning to actuate different hand-scripted behaviours. Every behaviour learns when it should be active. It is done by evaluating the correlation

between positive and negative feedback and the behaviour activation status; and by calculating the probability that the behaviour becomes active. It also evaluates the correlation between some conditions, the behaviour activation status and the positive or negative feedback. The algorithm worked successfully on a simple system having binary inputs and outputs (ON-OFF, active, inactive, move-leg-forward, move-leg-backward, etc.) and a single goal. It has not been demonstrated on systems acting in complex environments with continuous state and action spaces. It would be challenging to use it on a robot having multiple goals and navigating in a complex world. Not every state of a mobile robot can be described easily by on or off, active or inactive, but it shows that algorithmic simplicity could be key in the solution of mobile robot tasks.

Several authors proposed using passive learning as a first learning stage by supplying a control policy to accelerate the learning process. They suggested running the robots remotely or using hand-scripted programs. This method incorporates human experience into the learning process and reduces the time spent exploring the world randomly. Also, some proposed using these online explorations runs to then proceed to offline policy learning. It reduces the amount of real robot runs required to learn the correct policy. The passive learning technique is a promising approach to train the STRV. The human approach could be incorporated into the learning process by driving the robot with a joystick. The information gathered could then be used for online training of the system to improve the control policy efficiency. Finally, using a simulation environment, it could be possible to visualize the effectiveness of the resulting policy in navigating the robot within the world and negotiating obstacles. All those steps would accelerate the learning process and reduce the amount of real runs to obtain a correct control policy.

Reinforcement learning has been used in mobile robot navigation by numerous researchers. Some interesting algorithms have been presented in this literature review. In complex terrains, robot mobility could be improved by learning the appropriate behaviours for the situations encountered. Reinforcement learning is an attractive concept since it can learn in real-time based on rewards and penalties. A lot of work still has to be done to provide robots with the ability to efficiently learn to behave adequately in complex environments.

References

[1] Steigerwald, B. (25 Feb. 2006). Shape-Shifting Robot Nanotech Swarms on Mars. (Online) National Aeronautics and Space Administration. http://www.nasa.gov/vision/universe/roboticexplorers/ants.html (3 Apr. 2006).

[2] Knight, W. (17 Nov. 2004). Shape-Shifting robot shows off its moves. (Online) NewScientist.com. http://www.newscientist.com/article.ns?id=dn6683 (3 Apr. 2006).

[3] Kamimura, A., Kurokawa, H., Yoshida, E., Murata, S., Tomita, K., and Kokaji, S. (2004). Distributed Adaptive Locomotion by a Modular Robotic System, M-TRAN II: From Local Adaptation to Global Coordinated Motion using CPG Controllers. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2370–2377.

[4] Iagnemma, K. and Dubowsky, S. (2004). Mobile Robots in Rough Terrain: Estimation, Motion Planning, and Control with Application to Planetary Rovers, Vol. 12 of *Springer tracts in Advanced Robotics (STAR) Series*.

[5] Kaelbling, L.P. (1993). Learning in Embedded Systems, The MIT Press.

[6] Grimsha, D. (2001). Definition of Agent. (Online) Ryerson University. http://www.ryerson.ca/ dgrimsha/courses/cps720/agentDef.html (3 Apr. 2006).

[7] Smart, W.D. and Kaelbling, L.P. (2002). Effective Reinforcement Learning for Mobile Robots. In *Proceedings of the International Conference on Robotics and Automation*, pp. 3404–3410. Piscataway, NJ: IEEE Press.

[8] Kaelbling, L.P., Littman, M.L., and Moore, A.W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

[9] Mitchell, T.M. (1997). Machine Learning, WBC McGraw-Hill.

[10] Smart, W.D. and Kaelbling, L.P. (2000). Practical Reinforcement Learning in Continuous Spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 903–910. San Francisco, CA: Morgan Kaugmann.

[11] Mataric, M.J. (1994). Reward Functions for Accelerated Learning. In *Proceedings of the eleventh International Conference on Machine Learning*, pp. 181–189. San Francisco, CA: Morgan Kaufmann.

[12] Gullapalli, V. (1992). Reinforcement learning and its application to control. Ph.D. thesis. University of Massachusetts.

[13] Zavlangas, P.G., Tzafestas, S.G., and Althoefer, K. (2000). Fuzzy obstacle avoidance and navigation for omnidirectional mobile robots. In *Proceedings of the third European Symposium on Intelligent Techniques*, pp. 375–382.

[14] Saffiotti, A. (7 Aug. 1997). Fuzzy logic in Autonomous Robot Navigation: a case study. (Online) University of Orebro. http://www.aass.oru.sa/Agora/FLAR/HFC (3 Apr. 2006).

[15] Yung, N. H. C. and Ye, C. (1996). Self-learning Fuzzy Navigation of Mobile Vehicle. In *Proceedings of the International Conference on Signal Processing*, pp. 1465–1468. Beijing, China: ICSP 1996.

[16] Bakker, B., Zhumatiy, V., Gruener, G., and Schmidhuber, J. (2003). A Robot that Reinforcement-Learns to Identify and Memorize Important Previous Observations. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 430–435. Las Vegas: IROS 2003.

[17] Bakker, B., Linaker, F., and Schmidhuber, J. (2002). Reinforcement Learning in Partially Observable Mobile Robot Domains Using Unsupervised Event Extraction. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 938–943. Lausanne: IROS 2002.

[18] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. In *Neural Computation*, Vol. 9, pp. 1735–1780.

[19] Bakker, B.. Reinforcement Learning with Long Short-Term Memory. In Dietterich, T.G., Becker, S., and Ghahramani, Z., (Eds.), *Advances in Neural Information Processing Systems*, Mit press ed, Vol. 14, pp. 1475–1482. Cambridge, MA.

[20] Harmon, M.E. and Baird, L.C. (1996). Multi-player residual advantage learning with general function approximation. Technical Report. Wright-Patterson Air Force Base.

[21] Singh, S.P. (1992). Transfer of learning by Composing Solutions of Elemental Sequential Tasks. In *Machine Learning*, Vol. 8, pp. 323–340.

[22] Singh, S.P.. The Efficient Learning of Multiple Task Sequences. In Moody, J., Hanson, S., and Lippman, R., (Eds.), *Neural Information Processing Systems*, Vol. 4, pp. 251–258. San Mateo, CA: Morgan Kaufmann.

[23] Tham, C.K. and Prager, R.W. (1994). A Modular Q-Learning Architecture for Manipulator Task Decomposition. In *Proceedings of the eleventh International Conference on Machine Learning*, pp. 309–317. New Brunswick, NJ: Morgan Kaufmann.

[24] Lin, L.-J. (1993). Scaling-up reinforcement learning for robot control. In *Proceedings of the 10th International Conference on Machine Learning*, pp. 182–189. Amherst, MA: Morgan Kaufmann.

[25] Humphrys, M. (1997). Action Selection methods using Reinforcement learning. Ph.D. thesis. University of Cambridge.

[26] Brooks, R.A. (1986). A robust layered control system for mobile robot. *IEEE Journal of Robotics and Automation*, *2*(1), 14–23.

[27] Maes, P. and Brooks, R.A. (1990). Learning to Coordinate Behaviors. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pp. 796–802. San Mateo, CA: Morgan Kaufman.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

   Defence R&D Canada – Suffield
   Box 4000, Station Main, Medicine Hat, Alberta,
   Canada T1A 8K6

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).

   UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

   Reinforcement learning in mobile robot navigation

4. AUTHORS (last name, first name, middle initial)

   Vincent, I.

5. DATE OF PUBLICATION (month and year of publication of document)	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc).	6b. NO. OF REFS (total cited in document)
December 2006	38	27

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

   Technical Memorandum

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).

   Defence R&D Canada – Suffield
   Box 4000, Station Main, Medicine Hat, Alberta, Canada T1A 8K6

9a. PROJECT NO. (the applicable research and development project number under which the document was written. Specify whether project).	9b. GRANT OR CONTRACT NO. (if appropriate, the applicable number under which the document was written).

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique.)	10b. OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Suffield TM 2006-117	

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

    (X) Unlimited distribution
    ( ) Defence departments and defence contractors; further distribution only as approved
    ( ) Defence departments and Canadian defence contractors; further distribution only as approved
    ( ) Government departments and agencies; further distribution only as approved
    ( ) Defence departments; further distribution only as approved
    ( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

    Unlimited

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Robotics is gaining popularity in military operations to facilitate soldier tasks and decrease their exposure to dangerous situations. Researchers are currently working on autonomous and semi-autonomous robots to provide soldiers with more intelligent robotic vehicles. At DRDC Suffield, the Autonomous Intelligent Systems Section is building an expertise in intelligent mobility for unmanned ground vehicles by developing robotic platforms that autonomously generate useful locomotive behaviours. One objective of the group is to solve the problem of navigability of shape-shifting mobile robot platforms. Learning to choose the appropriate geometric configuration with respect to the environment is a promising solution. This document presents a literature review emphasizing reinforcement learning in mobile robot navigation. A variety of algorithms are examined in detail such as Q-learning, HEDGER, SRV, RL-LSTM, CQ-L, HQ-L and W-learning. Finally, the applicability of those algorithms to the problem of shape-shifting mobile robot navigation is discussed.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

Machine learning
Mobile robot
Navigation