26

A Robot Visual Homing Model that Traverses Conjugate Gradient TD to a Variable λ TD and uses Radial Basis Features.

Abdulrahman Altahhan
Yarmouk Private University
Syria

1. Introduction

The term 'homing' refers to the ability of an agent – either animal or robot - to find a known goal location. It is often used in the context of animal behaviour, for example when a bird or mammal returns 'home' after foraging for food, or when a bee returns to its hive. Visual homing, as the expression suggests, is the act of finding a home location using vision. Generally it is performed by comparing the image currently in view with 'snapshot' images of the home stored in the memory of the agent. A movement decision is then taken to try and match the current and snapshot images (Nehmzow 2000).

A skill that plays a critical role in achieving robot autonomy is the ability to learn to operate in previously unknown environments (Arkin 1998; Murphy 2000; Nehmzow 2000). Furthermore, learning to home in unknown environments is a particularly desirable capability. If the process was automated and straightforward to apply, it could be used to enable a robot to reach any location in any environment, and potentially replace many existing computationally intensive navigation and localisation algorithms. Numerous models have been proposed in the literature to allow mobile robots to navigate and home in a wide range of environments. Some focus on learning (Kaelbling, Littman et al. 1998; Nehmzow 2000; Asadpour and Siegwart 2004; Szenher 2005; Vardy and Moller 2005), whilst others focus on the successful application of a model or algorithm for a specific environment and ignore the learning problem (Simmons and Koenig 1995; Thrun 2000.; Tomatis, Nourbakhsh et al. 2001).

Some robotic approaches borrow conceptual mechanisms from animal homing and navigation strategies described in neuroscience or cognition literature (Anderson 1977; Cartwright and Collett 1987). Algorithms based on the snapshot model use various strategies for finding features within images and establishing correspondence between them in order to determine home direction (Cartwright and Collett 1987; Weber, Venkatesh et al. 1999; Vardy and Moller 2005). Block matching, for example, takes a block of pixels from the current view image and searches for the best matching block in stored images within a fixed search radius (Vardy and Oppacher 2005).

Most robot homing models proposed in the literature have the limitations of either depending upon landmarks (Argyros, Bekris et al. 2001; Weber, Wermter et al. 2004; Muse, Weber et al. 2006), which makes them environment-specific, or requiring pre-processing stages, in order for them to learn or perform the task (Szenher 2005; Vardy 2006). These assumptions restrict the employability of such models in a useful and practical way. Moreover, new findings in cognition suggest that humans are able to home in the absence of feature-based landmark information (Gillner, Weiß et al. 2008). This biological evidence suggests that in principle at least the limitations described are unnecessarily imposed on existing models.

This chapter describes a new visual homing model that does not require either landmarks or pre-processing stages. To eliminate the landmark requirement whole image measures of current views and stored snapshots are used in the model (Ulrich and Nourbakhsh 2000); object recognition is not necessary to identify a location, or to perform navigation, homing or localization tasks. To eliminate the pre-processing requirement it was necessary to employ a general learning process capable of capturing the specific characteristics of any environment, without the need to customise the model architecture.

Reinforcement learning (RL) provides such a capability and this coupled with visual homing based on a whole image measure forms the first novelty of the work.

RL has been used previously in robot navigation and control, including several models inspired by biological findings (Weber, Wermter et al. 2004; Sheynikhovich, Chavarriaga et al. 2005). However some of those models lack the generality and/or practicality, and some are restricted to their environment; the model proposed by (Weber, Wermter et al. 2004; Muse, Weber et al. 2006; Weber, Muse et al. 2006), for example, depends on object recognition of a landmark in the environment to achieve the task. Therefore, the aim of the work described in this chapter was to exploit the capability of RL as much as possible by general model design, as well as by using a whole image measure. RL advocates a general learning approach that avoids human intervention of supervised learning and, unlike unsupervised learning, has a specific problem-related target that should be met. Furthermore, since RL deals with reward and punishment it has strong ties with biological systems, making it suitable for the homing problem. Whilst environment-dynamics or map-building may be necessary for more complex or interactive forms of navigation or localization, visual homing based on model-free learning can offer an adaptive form of local homing. In addition, although the immediate execution of model-based navigation can be successful (Thrun, Liu et al. 2004; Thrun, Burgard et al. 2005), RL techniques have the advantage of being model-free i.e. no knowledge needed about the environment. The agent learns the task by learning the best policy that allows it to collect the largest sum of rewards from its environment according to the environment dynamics.

The second novelty of this work is related to enhancing the performance of the an existing RL method. Reinforcement learning with function approximation has been shown in some cases to learn slowly (Bhatnagar, Sutton et al. 2007). Bootstrapping methods like temporal difference (TD) (Sutton 1988) although was proved to be faster than other RL methods, such as residual gradient established by Baird (Baird 1995), it can still be slow (Schoknecht and Merke 2003). Slowness in TD methods can occur due to different reasons. The frequent cause is when the state space is big, high-dimensional or continuous. In this case, it is hard to maintain the value of each state in a tabular form. Even when the state space is approximated in some way, using artificial neural networks (ANN) for example, the learning process can become slow because it is still difficult to generalize in such huge spaces. In order for TD to converge when used for prediction, all states should be visited frequently enough. For large state spaces this means that convergence may involve many steps and will become slow.

Numerical techniques have been used with RL methods to speed up its performance. For example, (Ziv and Shimkin 2005) used a multi-grid framework which is originated in numerical analysis to enhance the iterative solution of linear equations. Whilst, other attempt to speed up RL method performance in multi-agent scenario, (Zhang, Abdallah et al. 2008), by using a supervised approach combined with RL to enhance the model performance. TD can be speed up by using it with other gradient types. In (Bhatnagar, Sutton et al. 2007), for example, TD along with the natural gradient has been used to boost learning.

(Falas and Stafylopatis 2001; Falas and Stafylopatis 2002) have used conjugate gradient with TD. Their early experiments confirmed that using such a combination can enhance the performance of TD. Nevertheless, no formal theoretical study has been conducted which disclose the intrinsic properties of such a combination. The present work is an attempt to fill this gap. It uncover an interseting property of combining TD method with the conjugate gradient which simplifies the implementation of the conjugate TD.

The chapter is structured as follows. Firstly an overview of TD and function approximation is presented, followed by the induction of the TD-conj learning and its novel equivalency property. Then a detail describtion of the novel visual homing model and its components is presneted. The results of extensive simulations and experimental comparisons are shown, followed by conclusions and recommendations for further work.

2. TD and function approximation

When function approximation techniques are used to learn a parametric estimate of the value function $V^{\pi}(s)$, $V_t(s_t)$ should be expressed in terms of some parameters $\vec{\theta}_t$. The mean squared error performance function can be used to drive the learning process:

$$F_t = MSE(\vec{\theta}_t) = \sum_{s \in S} pr(s) \left[V^{\pi}(s) - V_t(s) \right]^2 \tag{1}$$

pr is a probability distribution weighting the errors $Er_t^2(s)$ of each state, and expresses the fact that better estimates should be obtained for more frequent states where:

$$Er_t^2(s) = \left[V^{\pi}(s) - V_t(s)\right]^2 \tag{2}$$

The function F_t needs to be minimized in order to find an optimal solution $\vec{\theta}^*$ that best approximates the value function. For on-policy learning if the sample trajectories are being drawn according to *pr* through real or simulated experience, then the update rule can be written as:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \frac{1}{2}\alpha_t \vec{d}_t \tag{3}$$

$\vec{d}_t$ is a vector that drives the search for $\vec{\theta}^*$ in the direction that minimizes the error function $Er_t^2(s)$, and $0 < \alpha_t \le 1$ is a step size. Normally going opposite to the gradient of a function leads the way to its local minimum. The gradient $\vec{g}_t$ of the error $Er_t^2(s)$ can be written as:

$$\vec{g}_t = \nabla_{\vec{\theta}_t} Er_t^2(s_t) = 2\left[V^{\pi}(s_t) - V_t(s_t)\right] \cdot \nabla_{\vec{\theta}_t} V_t(s_t) \tag{4}$$

Therefore, when $\vec{d}_t$ is directed opposite to $\vec{g}_t$, i.e. $\vec{d}_t = -\vec{g}_t$, we get the gradient descent update rule:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t \left[V^{\pi}(s_t) - V_t(s_t)\right] \cdot \nabla_{\vec{\theta}_t} V_t(s_t) \tag{5}$$

It should be noted that this rule allows us to obtain an estimate of the value function through simulated or real experience in a supervised learning (SL) fashion. However, even for such samples the value function V^{π} can be hard to be known in a priori. If the target value function V^{π} of policy π is not available, and instead some other approximation of it is, then an approximated form of rule (5) can be realized. For example, replacing R_t by V^{π} produces the Monte Carlo update $\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t \left[R_t - V_t(s_t)\right] \cdot \nabla_{\vec{\theta}_t} V_t(s_t)$. By its definition R_t is an unbiased estimate for $V^{\pi}(s_t)$, hence this rule is guaranteed to converge to a local minima. However, this rule requires waiting until the end of the task to obtain the quantity R_t to perform the update. This demand can be highly restrictive for the practical application of such rule. On the other hand, if the n-step return $R_t^{(n)} = \sum_{i=1}^{n} \gamma^{i-1} r_{t+i} + \gamma^n V_t(s_{t+n})$ is used to approximate $V^{\pi}(s_t)$, then from (5) we obtain the rule $\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t \left[R_t^{(n)} - V_t(s_t)\right] \cdot \nabla_{\vec{\theta}_t} V_t(s_t)$ which is less restrictive and of more practical interest than rule (5) since it requires only to wait n steps to obtain $R_t^{(n)}$. Likewise, any averaged mixture of $R_t^{(n)}$ (such as $\frac{1}{2} R_t^{(1)} + \frac{1}{2} R_t^{(3)}$) can be used, as long as the coefficients sum up to 1. An important example of such averages is the sum $R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$ which also can be used to get the update rule:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t \left[R_t^{\lambda} - V_t(s_t)\right] \cdot \nabla_{\vec{\theta}_t} V_t(s_t) \tag{6}$$

Unfortunately, however, $R_t^{(n)}$ (and any of its averages including R_t^{λ}) is a biased approximation of $V^{\pi}(s_t)$ for the very reason that makes it practical (which is not waiting until the end of the task to obtain $V^{\pi}(s_t)$). Hence rule (6) does not necessary converge to a local optimum solution $\vec{\theta}^*$ of the error function F_t. The resultant update rule (6) is in fact the forward view of the TD(λ) method where no guarantee of reaching $\vec{\theta}^*$ immediately follows. Instead, under some conditions, and when linear function approximation are used, then the former rule is guaranteed to converge to a solution $\vec{\theta}_{\infty}$ that satisfies (Tsitsiklis and Van Roy 1997) $MSE^{\frac{1}{2}}(\vec{\theta}_{\infty}) \le \frac{1-\gamma\lambda}{1-\gamma} MSE^{\frac{1}{2}}(\vec{\theta}^*)$.

The theoretical forward view of the TD updates involves the quantity R_t^{λ} which is, in practice, still hard to be available because it needs to look many steps ahead in the future. Therefore, it can be replaced by the mechanistic backward view involving eligibility traces. It can be proved that both updates are equivalent for the off-line learning case (Sutton and Barto 1998) even when λ varies from one step to the other as long as $\lambda \in [0,1]$. The update rules of TD with eligibility traces, denoted as TD(λ) are:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t \delta_t \cdot \vec{e}_t \tag{7}$$

$$\vec{e}_t = \gamma\lambda \vec{e}_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t) \tag{8}$$

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t) \tag{9}$$

It can be realized that the forward and backward rules become identical for TD(0). In addition, the gradient can be approximated as:

$$\vec{g}_t = 2\delta_t \cdot \nabla_{\vec{\theta}_t} V_t(s_t) \tag{10}$$

If linear neural network is used to approximate $V_t(s_t)$, then it can be written as $V_t(s_t) = \vec{\phi}_t^T \vec{\theta}_t = \vec{\theta}_t^T \vec{\phi}_t$. In this case, we obtain the update rule:

$$\vec{e}_t = \gamma\lambda \vec{e}_{t-1} + \vec{\phi}_t \tag{11}$$

It should be noted that all of the former rules starting from (4) depend on the gradient decent update. When the quantity $V^\pi(s_t)$ was replaced by some approximation the rules became impure gradient decent rules. Nevertheless, such rules can still called gradient decent update rules since they are derived according to it. Rules that uses its own approximation of $V^\pi(s_t)$ are called bootstrapping rules. In particular, TD(0) update is a bootstrapping method since it uses the term $r_{t+1} + \gamma V_t(s_{t+1})$ which involves its own approximation of the value of the next state to approximate $V^\pi(s_t)$. In addition, the gradient $\vec{g}_t$ can be approximated in different ways. For example, if we approximate $V^\pi(s_t)$ by $r_{t+1} + \gamma V_t(s_{t+1})$ first then calculate the gradient we get the residual gradient temporal difference, which in turn can be combined with TD update in a weight averaged fashion to get the residual TD (Baird 1995).

3. TD and conjugate gradient function approximation

3.1 Conjugate gradient extension of TD

We turn our attention now for an extension of TD(λ) learning using function approximation. We will direct the search for the optimal points of the error function $Er_t^2(s)$ along the conjugate direction instead of the gradient direction. By doing so an increase in the performance is expected. In fact, more precisely a decrease of the number of steps to reach optimality is expected. This is especially true for cases where the number of distinctive eigenvalues of the matrix H (matrix of second derivatives of the performance function) is less than n the number of parameters $\vec{\theta}$. To direct the search along the conjugate gradient direction, $\vec{p}$ should be constructed as follows:

$$\vec{p}_t = -\vec{g}_t + \beta_t \vec{p}_{t-1} \tag{12}$$

$\vec{p}_0$ is initiated to the gradient of the error (Hagan, Demuth et al. 1996; Nocedal and Wright 2006); $\vec{p}_0 = -\vec{g}_0$. Rule (12) ensures that all $\vec{p}_t \ \forall t$ are orthogonal to $\Delta\vec{g}_{t-1} = \vec{g}_t - \vec{g}_{t-1}$. This can be realized by choosing the scalar β_t to satisfy the orthogonality condition:

$$\Delta\vec{g}_{t-1} \cdot \vec{p}_t = 0 \Rightarrow \Delta\vec{g}_{t-1}^T \vec{p}_t = 0 \Rightarrow \Delta\vec{g}_{t-1}^T(-\vec{g}_t + \beta_t \vec{p}_{t-1}) = 0 \Rightarrow \beta_t = \frac{\Delta\vec{g}_{t-1}^T \vec{g}_t}{\Delta\vec{g}_{t-1}^T \vec{p}_{t-1}} \tag{13}$$

In fact, the scalar β_t can be chosen in different ways that should produce equivalent results for the quadratic error functions (Hagan, Demuth et al. 1996), the most common choices are:

$$\beta_t^{(HS)} = \frac{\Delta\vec{g}_{t-1}^T \vec{g}_t}{\Delta\vec{g}_{t-1}^T \vec{p}_{t-1}} \ (14), \quad \beta_t^{(FR)} = \frac{\vec{g}_t^T \vec{g}_t}{\vec{g}_{t-1}^T \vec{g}_{t-1}} \qquad (15), \quad \beta_t^{(PR)} = \frac{\Delta\vec{g}_{t-1}^T \vec{g}_t}{\vec{g}_{t-1}^T \vec{g}_{t-1}} \tag{16}$$

due to Hestenes and Steifel, Fletcher and Reeves, and Polak and Ribiere respectively.
From equation (4) the conjugate gradient rule (12) can be rewritten as follows:

$$\vec{p}_t = 2\left[V^\pi(s_t) - V_t(s_t)\right] \cdot \nabla_{\vec{\theta}_t} V_t(s_t) + \beta_t \vec{p}_{t-1} \tag{17}$$

By substituting in (3) we obtain the pure conjugate gradient general update rule:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \frac{1}{2}\alpha_t \left[2\left[V^\pi(s_t) - V_t(s_t)\right]\nabla_{\vec{\theta}_t} V_t(s_t) + \beta_t \vec{p}_{t-1}\right] \tag{18}$$

3.2 Forward view of conjugate gradient TD

Similar to TD(λ) update rule (6), we can approximate the quantity $V^\pi(s_t)$ by R_t^λ, which does not guarantee convergence, because it is not unbiased (for λ < 1), nevertheless it is more practical. Hence, we get the theoretical forward view of TD-conj(λ); the TD(λ) conjugate gradient update rules:

$$\vec{p}_t = 2\left[R_t^\lambda - V_t(s_t)\right]\cdot\nabla_{\vec{\theta}_t}V_t(s_t) + \beta_t\vec{p}_{t-1} \tag{19}$$

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t\left[\left[R_t^\lambda - V_t(s_t)\right]\nabla_{\vec{\theta}_t}V_t(s_t) + \frac{1}{2}\beta_t\vec{p}_{t-1}\right] \tag{20}$$

If the estimate $r_{t+1} + \gamma V_t(s_{t+1})$ is used to estimate $V^\pi(s_t)$ (as in TD(0)), then we can obtain the TD-conj(0) update rules; where rules (19) and (20) are estimated as:

$$\vec{p}_t = 2\delta_t\cdot\nabla_{\vec{\theta}_t}V_t(s_t) + \beta_t\vec{p}_{t-1} \tag{21}$$

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t\left[\delta_t\nabla_{\vec{\theta}_t}V_t(s_t) + \frac{1}{2}\beta_t\vec{p}_{t-1}\right] \tag{22}$$

It should be noted again that those rules are not pure conjugate gradient but nevertheless we call them as such since they are derived according to the conjugate gradient rules.

3.3 Equivalency of TD(λt≠0) and TD-conj(λ=0)

Theorem 1:

TD-conj(0) is equivalent to a special class of TD(λ_t) that is denoted as TD($\lambda_t^{(conj)}$), under the condition: $0 \le \lambda_t^{(conj)} = \frac{\beta_t}{\gamma}\frac{\delta_{t-1}}{\delta_t} \le 1;\ \forall t$, regardless of the approximation used. The equivalency is denoted as TD-conj(0) ≡ TD($\lambda_t^{(conj)}$), and the bound condition is called the equivalency condition.

Proof:

We will proof that TD-conj(0) is equivalent to a backward view of a certain class of TD(λt), denoted as TD($\lambda_t^{(conj)}$). Hence, by the virtue of the equivalency of the backward and forward views of all TD(λ_t) for the off-line case, the theorem follows. For the on-line case the equivalency is restricted to the backward view of TD($\lambda_t^{(conj)}$).

The update rule (22) can be rewritten in the following form:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t\delta_t\left[\nabla_{\vec{\theta}_t}V_t(s_t) + \frac{\beta_t}{2\delta_t}\vec{p}_{t-1}\right] \tag{23}$$

where it is assumed that $\delta_t \ne 0$ because otherwise it means that we reached an equilibrium point for $V_t(s_t)$, meaning the rule has converged and there is no need to apply any learning rule any more. Now we introduce the conjugate eligibility traces vector $\vec{e}_t^{(conj)}$ that is defined as follows:

$$\vec{e}_t^{(conj)} = \frac{1}{2\delta_t}\vec{p}_t \tag{24}$$

By substituting (21) in (24) we have that

$$\vec{e}_t^{(conj)} = \nabla_{\vec{\theta}_t}V_t(s_t) + \frac{\beta_t}{2\delta_t}\vec{p}_{t-1} \tag{25}$$

From (25) we proceed in two directions. First, by substituting (25) in (23) we obtain an update rule identical to rule (11):

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t\delta_t\vec{e}_t^{(conj)} \tag{26}$$

Second, from (24) we have that:

$$\vec{p}_{t-1} = 2\delta_{t-1}\vec{e}_{t-1}^{(conj)} \tag{27}$$

Hence, by substituting (27) in (25) we obtain:

$$\vec{e}_t^{(conj)} = \nabla_{\vec{\theta}_t}V_t(s_t) + \frac{\delta_{t-1}}{\delta_t}\beta_t\vec{e}_{t-1}^{(conj)} \tag{28}$$

By conveniently defining:

$$\gamma\lambda_t^{(conj)} = \beta_t\frac{\delta_{t-1}}{\delta_t} \tag{29}$$

we acquire an update rule for the conjugate eligibility traces $\vec{e}_t^{(conj)}$ that is similar to rule (8):

$$\vec{e}_t^{(conj)} = \gamma\lambda_t^{(conj)}\vec{e}_{t-1}^{(conj)} + \nabla_{\vec{\theta}_t}V_t(s_t) \tag{30}$$

Rules (26) and (30) are almost identical to rules (8) and (9) except that λ is variable in (29). Hence, they show that TD-conj(0) method can be equivalent to a backward update of TD(λ) method with a variable λ. In addition, rule (29) establishes a canonical way of varying λ; where we have:

$$\lambda_t^{(conj)} = \frac{\beta_t}{\gamma}\frac{\delta_{t-1}}{\delta_t} \tag{31}$$

The only restriction we have is that there is no immediate guarantee that $\lambda_t^{(conj)} \in [0,1]$. Hence, the condition for full equivalency is that $\lambda_t^{(conj)}$ satisfies:

$$0 \le \lambda_t^{(conj)} = \frac{\beta_t}{\gamma}\frac{\delta_{t-1}}{\delta_t} \le 1 \tag{32}$$

According to (31) and by substituting (14),(15) and(16) we obtain the following different ways of calculating $\lambda_t^{(conj)}$:

$$^{1}\lambda_t^{(conj)} = \frac{\left(\delta_t \cdot \nabla_{\vec{\theta}_t} V_t(s_t) - \delta_{t-1} \cdot \nabla_{\vec{\theta}_{t-1}} V_{t-1}(s_{t-1})\right)^T \nabla_{\vec{\theta}_t} V_t(s_t)}{\gamma\left(\delta_t \cdot \nabla_{\vec{\theta}_t} V_t(s_t) - \delta_{t-1} \cdot \nabla_{\vec{\theta}_{t-1}} V_{t-1}(s_{t-1})\right)^T \vec{e}_{t-1}^{(conj)}} \tag{33}$$

$$^{2}\lambda_t^{(conj)} = \frac{\delta_t \left\| \nabla_{\vec{\theta}_t} V_t(s_t)_t \right\|^2}{\gamma \delta_{t-1} \left\| \nabla_{\vec{\theta}_{t-1}} V_{t-1}(s_{t-1}) \right\|^2} \tag{34}$$

$$^{3}\lambda_t^{(conj)} = \frac{\left(\delta_t \cdot \nabla_{\vec{\theta}_t} V_t(s_t) - \delta_{t-1} \cdot \nabla_{\vec{\theta}_{t-1}} V_{t-1}(s_{t-1})\right)^T \nabla_{\vec{\theta}_t} V_t(s_t)}{\gamma \delta_{t-1} \left\| \nabla_{\vec{\theta}_{t-1}} V_{t-1}(s_{t-1}) \right\|^2} \tag{35}$$

which proves our theorem □.

There are few things to be realized from Theorem 1:

$\lambda_t^{(conj)}$ should be viewed as a more general form of λ which can magnify or shrink the trace according to how much it has confidence in its estimation. TD with variable λ has not been studied before (Sutton and Barto 1998), and $\lambda_t^{(conj)}$ gives for the first time a canonical way to vary λ depending on conjugate gradient directions.

From (31) it can be realized that both δ_t and δ_{t-1} are involved in the calculations of the eligibility traces. This means that the division cancels the direct effect of an error δ and leaves the relative rate-of-changes between consequent steps of this error to play the big role in changing $\lambda_t^{(conj)}$ according to (31).

Since γ satisfies $\gamma \in [0,1]$ and $\lambda_t^{(conj)}$ should satisfy that $\lambda_t^{(conj)} \in [0,1]$, so as the term $\gamma\lambda_t^{(conj)}$ should satisfy $0 \le \gamma\lambda_t^{(conj)} \le 1$. Therefore, the (32) condition can be made more succinct:

$$0 \le \beta_t \frac{\delta_{t-1}}{\delta_t} \le \gamma \le 1 \tag{36}$$

The initial eligibility trace is:

$$\vec{e}_0^{(conj)} = \frac{1}{2\delta_0}\vec{p}_0 = \frac{1}{2\delta_0} 2\delta_0 \nabla_{\vec{\theta}_t} V_0(s_0) \Rightarrow \qquad \vec{e}_0^{(conj)} = \nabla_{\vec{\theta}_t} V_0(s_0) \tag{37}$$

From an application point of view, it suffices for $\lambda_t^{(conj)}$ value to be forced to this condition whenever its value goes beyond 1 or less than 0:

$$if\,(\lambda_t^{(conj)} > 1) \Rightarrow \lambda_t^{(conj)} \leftarrow 1, \qquad if\,(\lambda_t^{(conj)} < 0) \Rightarrow \lambda_t^{(conj)} \leftarrow 0 \tag{38}$$

If we approximated $V_t(s_t)$ by a linear approximation $V_t(s_t) = \vec{\theta}_t^T \vec{\phi}_t$ then: $\nabla_{\vec{\theta}_t} V_t(s_t) = \vec{\phi}_t$,in this case we have from (30) that:

$$\vec{e}_t^{(conj)} = \gamma\lambda_t^{(conj)} \vec{e}_{t-1}^{(conj)} + \vec{\phi}_t \tag{39}$$

λ can be defined by substituting $\nabla_{\vec{\theta}_t} V_t(s_t) = \vec{\phi}_t$ in (33), (34) and (35) respectively as follows:

$$^{1}\lambda_t^{(conj)} = \frac{\left(\delta_t\vec{\phi}_t - \delta_{t-1}\vec{\phi}_{t-1}\right)^T \vec{\phi}_t}{\left(\delta_t\vec{\phi}_t - \delta_{t-1}\vec{\phi}_{t-1}\right)^T \vec{e}_{t-1}}, \quad ^{2}\lambda_t^{(conj)} = \frac{\delta_t \left\|\vec{\phi}_t\right\|^2}{\delta_{t-1}\left\|\vec{\phi}_{t-1}\right\|^2}, \quad ^{3}\lambda_t^{(conj)} = \frac{\left(\delta_t\vec{\phi}_t - \delta_{t-1}\vec{\phi}_{t-1}\right)^T \vec{\phi}_t}{\delta_{t-1}\left\|\vec{\phi}_{t-1}\right\|^2} \tag{40}$$

Any method that depends on TD updates such as Sarsa or Q-learning can take advantage of these new findings and use the new update rules of TD-conj(0).This concludes our study of the properties of TD-conj(0) method and we move next to the model.

4 The visual homing Sarsa-conj(0) model

For visual homing it is assumed that the image at each time step represents the current state, and the state space S is the set of all images, or views, that can be taken for any location (with specific orientation) in the environment. This complex state space has two problems. Firstly, each state is of high dimensionality, i.e. it is represented by a large number of pixels. Secondly, the state space is huge, and a policy cannot be learned directly for each state. Instead, a feature representation of the states is used to reduce the high-dimensionality of the state space and to gain the advantages of coding that allows a parameterized representation to be used for the value function (Stone, Sutton et al. 2005). In turn, parameterization permits learning a general value function representation that can easily accommodate for new unvisited states by generalization. Eventually, this helps to solve the second problem of having to deal with a huge state space.

The feature representation can reduce the high-dimensionality problem simply by reducing the number of components needed to represent the views. Hence, reducing dimensionality is normally carried out at the cost of less distinctiveness for states belonging to a huge space. Therefore, the features representation of the state space, when successful, strikes a good balance between distinctiveness of states and reduced dimensionality. This assumption is of importance towards the realization of any RL model with a high-dimensional states problem.

4.1 State representation and home information

One representation that maintains an acceptable level of distinctiveness and reduces the high-dimensionality of images is the histogram. A histogram of an image is a vector of components, each of which contains the number of pixels that belong to a certain range of intensity values. The significance of histograms is that they map a large two-dimensional matrix to a smaller one-dimensional vector. This effectively encodes the input state space into a coarser feature space. Therefore, if the RGB (Red, Green, and Blue) representation of colour is used for an image the histogram of each colour channel is a vector of components, each of which is the number of pixels that lie in the component's interval. The interval each component represents is called the bin, and according to a pre-specified bin size of the range of the pixel values, a pre-specified number of bins will be obtained.

A histogram does not preserve the distinctiveness of the image, i.e. two different images can have the same histogram, especially when low granularity bin intervals are chosen. Nevertheless, histograms have been found to be widely acceptable and useful in image processing and image retrieval applications (Rubner and et al. 2000). Other representations exist, such as the one given by the Earth Mover's Distance (Rubner and et al. 2000). However, such mapping is not necessary for the problem here since the model will be dealing with a unified image dimension throughout its working life, because its images are captured by the same robot camera.

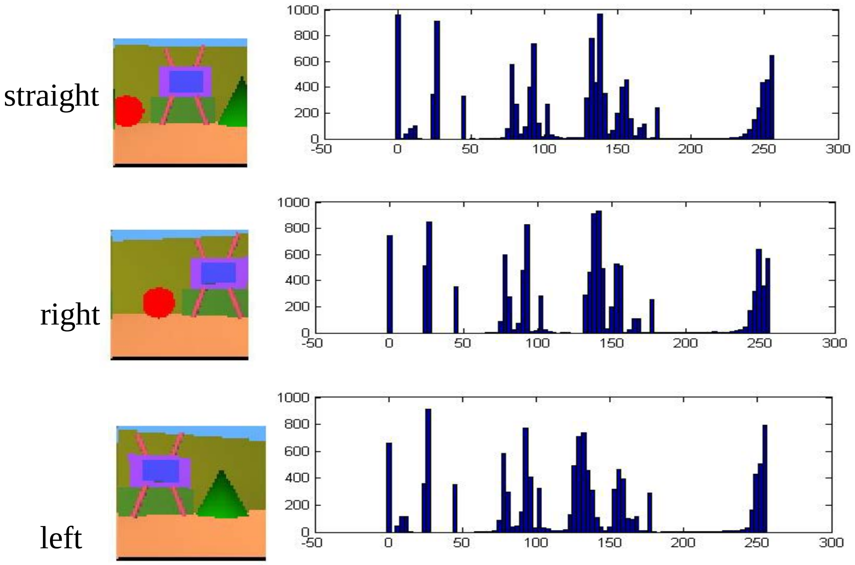

Fig. 1. Sample of a three-view representation taken from three different angles for a goal location with their associated histograms in a simulated environment.

The feature representation approach does not give a direct indication of the distance to the goal location. Although the assumption that the goal location is always in the robot's field of view will **not** be made, by comparing the current view with the goal view(s) the properties of distinctiveness, distance and orientation can be embodied to an extent in the RL model. Since the home location can be approached from different directions, the way it is represented should accommodate for those directions. Therefore, a home (or goal) location is defined by *m* snapshots called the stored views. A few snapshots (normally $m \geq 3$) of the home location are taken at the start of the learning stage, each from the same fixed distance but from a different angle. These snapshots define the home location and are the only information required to allow the agent to learn to reach its home location starting from any arbitrary position in the environment (including those from which it cannot see the home, i.e. the agent should be able to reach a hidden goal location). Fig. 1 shows a sample of a three-view representation of a goal location taken in a simulated environment.

4.2 Features vectors and radial basis representation

A histogram of each channel of the current view is taken and compared with those of the stored views through a radial basis function (RBF) component. This provides the features space $\Phi : S \rightarrow \Re^n$ representation (41) which is used with the Sarsa-conj algorithm, described later:

$$\phi_i(s_t(c, j)) = \exp\left(-\frac{(h_i(s_t(c)) - h_i(v(c, j)))^2}{2\hat{\sigma}_i^2} \right) \qquad (41)$$

Index t stands for the time step j for the jth stored view, and c is the index of the channel, where the RGB representation of images is used. Accordingly, $v(c, j)$ is the channel c image of the jth stored view, $h_i(v(c, j))$ is histogram bin i of image $v(c, j)$, and $h_i(s_t(c))$ is histogram bin i of channel c of the current (t) view. The number of bins will have an effect on the structure and richness of this representation and hence on the model. It should be noted that the radial basis feature extraction used here differs from the radial basis feature extraction used in (Tsitsiklis and Van Roy 1996). The difference is in the extraction process and not in the form. In their feature extraction, certain points s_i are normally chosen from the input space $\Re^n$ to construct a linear combination of radial basis functions. Those points in that representation are replaced in this work by the bins themselves.

Further, the variance of each bin will be substituted by a global average of the variances of those bins:

$$\hat{\sigma}_i^2 = (1/T-1)\sum_{t=1}^{T}\Delta h_i^2(t) \tag{42}$$

$$\Delta h_i^2(t) = \left(h_i(s_t(c)) - h_i(v(c,j))\right)^2 \tag{43}$$

where T is the total number of time steps. To normalize the feature representation the scaled histogram bins $h_i(s_t(c))/H$ are used, assuming that n is the number of features we have:

$$\sum_i^n h_i(v(c,j)) = \sum_i^n h_i(s_t(c)) = H \tag{44}$$

where it can be realized that H is a constant and is equal to the number of all pixels taken for a view. Hence, the final form of the feature calculation is:

$$\phi_i(s_t(c,j)) = \exp\left(-\frac{\left(h_i(s_t(c)) - h_i(v(c,j))\right)^2}{2H^2\hat{\sigma}^2}\right) \tag{45}$$

It should be noted that this feature representation has the advantage of being in the interval [0 1], which will be beneficial for the reasons discussed in the next section.

The feature vector of the current view (state) is a union of all of the features for each channel and each stored view, as follows:

$$\Phi(s_t) = \bigoplus_{j=1}^{m}\bigoplus_{c=1}^{3}\bigoplus_{i=1}^{B}\phi_i(s_t(c,j)) = (\phi_1,\ldots,\phi_i,\ldots,\phi_n) \tag{46}$$

where m is the number of stored views for the goal location, 3 channels are used, and B is the number of bins to be considered. Since an RGB image with values in the range of [0 255] for each pixel will be used, the dimension of the feature space is given by:

$$n = C\times B\times m = C\times(round(\frac{256}{b})+1)\times m \tag{47}$$

where b is the bin's size and $C=3$ is the number of channels. Different bin sizes give different dimensions, which in turn give different numbers of parameters $\vec{\theta}$ that will be used to approximate the value function.

4.3 NRB similarity measure and the termination condition

To measure the similarity between two images, the sum of all the Normalized Radial Basis (NRB) features defined above can be taken and then divided by the feature dimension. The resultant quantity is scaled to 1 and it expresses the overall belief that the two images are identical:

$$NRB(s_t) = \sum_{i=1}^{n}\bar{\phi}_i(s_t)/n \tag{48}$$

For simplicity the notation $NRB(s_t)$ and NRB_t will be used interchangeably. Other measures can be used (Rubner and et al. 2000). In previous work the Jeffery divergence measure was used (Altahhan, Burn et al. 2008). However the above simpler measure was adopted because it is computationally more efficient for the proposed model since it only requires an extra sum and a division operations. JDM has its own logarithmic calculation which cost additional computations.

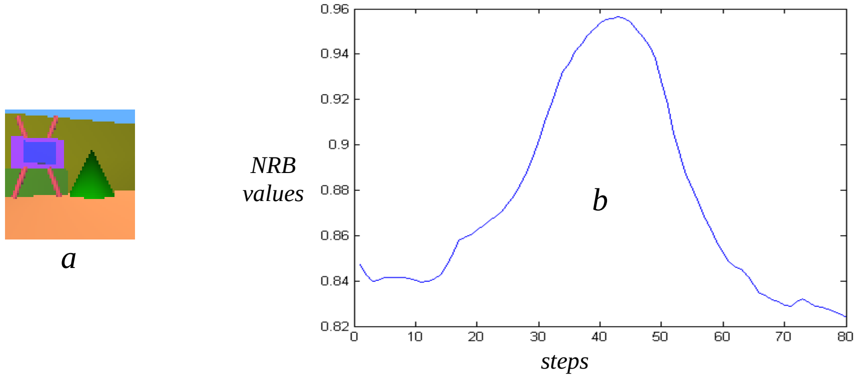

Fig. 2. (a): Current view of agent camera. (b) the plotted values of the normalized radial bases similarity measure of a sample π rotation.

Another benefit of NRB is that it is scaled to 1 – a uniform measure can be interpreted more intuitively. On the other hand, it is impractical to scale Jeffery Divergence Measure because although the maximum is known to be 0 there is no direct indication of the minimum. Figure 2 demonstrates the behaviour of NRB; the robot was placed in front of a goal location and the view was stored. The robot then has been let to rotate in its place from -90° (left) to +90° (right); in each time step the current view has been taken and compared with the stored view and their NRB value was plotted. As expected the normal distribution shape of those NRB values provide evidence for its suitability.

An episode describes the collective steps of an agent starting from any location and navigating in the environment until it reaches the home location. The agent is assumed to finish an episode and be in the home location (final state) if its similarity measure indicates with high certainty ψ_{upper} that its current view is similar to one of the stored views. This specifies the episode termination condition of the model.

$If\ NRB(s_t) \geq \psi_{upper} \quad \Rightarrow$ Terminate Episode

Similarly, the agent is assumed to be in the neighbourhood of the home location with the desired orientation $If\ NRB(s_t) \geq \psi_{lower}$ where $\psi_{upper} \geq \psi_{lower}$ this situation is called home-at-perspective and the interval $[\psi_{upper}, \psi_{lower}]$ is called the home-at-perspective confidence interval.home-at-perspectivehome-at-perspective

4.5 The action space

In order to avoid the complexity of dealing with a set of actions each with infinite resolution speed values (which in effect turns into an infinite number of actions), the two differential wheel speeds of the agent are assumed to be set to particular values, so that a set of three actions with fixed values is obtained. The set of actions is A = [Left_Forward, Right_Forward, Go_Forward]. The acceleration of continuous action space cannot be obtained using this limited set of actions. Nevertheless, by using actions with a small differential speed (i.e. small thrust rotation angle) the model can still get the effect of continuous rotation by repeating the same action as needed. This is done at the cost of more action steps.

A different set of actions than the limited one used here could be used to enhance the performance. For example, another three actions [Left_Forward, Right_Forward, Go_Forward] with double the speed could be added, although more training would be a normal requirement in this case. One can also add a layer to generalize towards other actions by enforcing a Gaussian activation around the selected action and fade it for other actions, as in (Lazaric, Restelli et al. 2007). In this work, however, the action set was kept to a minimum to concentrate on the effect of other components of the model.

5.6 The reward function

The reward function r depends on which similarity or dissimilarity function was used, and it consists of three parts:

$$r_{NRB} = \text{cost} + \Delta NRB_{t-1} + NRB_t \tag{49}$$

The main part is the cost, which is set to -1 for each step taken by the robot without reaching the home location. The other two parts are to augment the reward signal to provide better performance. They are:

Approaching the goal reward. This is the maximum increase in similarity between the current step and the previous step. This signal is called the differential similarity signal and it is defined as:

$$\Delta NRB_{t-1} = (NRB_t - NRB_{t-1}) \tag{50}$$

The Position signal, which is simply expressed by the current similarity NRB_t. Thus, as the current location differs less from the home location, this reward will increase. Hence, the reward can be rewritten in the following form:

$$r_{NRB} = \text{cost} + 2NRB_t - NRB_{t-1} \tag{51}$$

The two additional reward components above will be considered only if the similarity of t and t-1 steps are both beyond the threshold ψ_{lower} to ensure that home-at-perspective is satisfied in both steps. This threshold is empirically determined, and is introduced merely to enhance the performance.

5.7 Variable eligibility traces and update rule for TD-conj(0)

An eligibility trace constitutes a mechanism for temporal credit assignment. It marks the memory parameters associated with the action as being eligible for undergoing learning changes (Sutton and Barto 1998). For the visual homing application, the eligibility trace for the current action a is constructed from the feature vectors encountered so far. More specifically, it is the discounted sum of the feature vectors of the images that the robot has seen each time the same action a had been taken. The eligibility trace for other actions which have not been taken while in the current state is simply its previous trace but discounted, i.e. those actions are now less accredited, as demonstrated in the following equation.

$$\vec{\mathbf{e}}_t(a) \leftarrow \begin{cases} \gamma\lambda\vec{\mathbf{e}}_{t-1}(a) + \vec{\boldsymbol{\varphi}}(s_t) & \text{if } a = a_t \\ \gamma\lambda\vec{\mathbf{e}}_{t-1}(a) & \text{otherwise} \end{cases} \tag{52}$$

λ is the discount rate for eligibility traces $\vec{\mathbf{e}}_t$ and γ is the rewards discount rate . The eligibility trace components do not comply with the unit interval i.e. each component can be more than 1. The reward function also does not comply with the unit interval. The update rule that uses the eligibility trace and the episodically changed learning rate α_{ep} is as follows:

$$\vec{\boldsymbol{\theta}}(a_t) \leftarrow \vec{\boldsymbol{\theta}}(a_t) + \alpha_{ep} \cdot \vec{\mathbf{e}}_t(a_t) \cdot \delta_t \tag{53}$$

As it was shown above and in (Altahhan 2008) the conjugate gradient TD-conj(0) method is translated through an equivalency theorem into a TD(λ) method with variable λ denoted as TD($\lambda_t^{(conj)}$) with the condition that $0 \le \lambda_t^{(conj)} \le 1$. Therefore, to employ conjugate gradient TD, equation (52) can be applied to obtain the eligibility traces for TD-conj. The only difference is that λ is varying according to one of the following possible forms:

$$^{1}\lambda_t^{(conj)} = \frac{\left(\delta_t\vec{\phi}_t - \delta_{t-1}\vec{\phi}_{t-1}\right)^T \vec{\phi}_t}{\gamma\left(\delta_t\vec{\phi}_t - \delta_{t-1}\vec{\phi}_{t-1}\right)^T \vec{e}_{t-1}^{(conj)}},\quad {}^{2}\lambda_t^{(conj)} = \frac{\delta_t\left\|\vec{\phi}_t\right\|^2}{\gamma\delta_{t-1}\left\|\vec{\phi}_{t-1}\right\|^2}, \text{or } {}^{3}\lambda_t^{(conj)} = \frac{\left(\delta_t\vec{\phi}_t - \delta_{t-1}\vec{\phi}_{t-1}\right)^T \vec{\phi}_t}{\gamma\delta_{t-1}\left\|\vec{\phi}_{t-1}\right\|^2} \tag{54}$$

TD-conj(0) (and any algorithm that depends on it such as Sarsa-conj(0) (Altahhan 2008)) is a family of algorithms, not because its λ is changed automatically from one step to the other, but because λ can be varied using different types of formulae. Some of those formulae are outlined in (25) for linear function approximation.

In addition, those values of $\lambda_t^{(conj)}$ that do not satisfy $0 \le \lambda_t^{(conj)} \le 1$, can be forced according to the following:

$$if\,(\lambda_t^{(conj)} > 1) \Rightarrow \lambda_t^{(conj)} \leftarrow 1, \qquad if\,(\lambda_t^{(conj)} < 0) \Rightarrow \lambda_t^{(conj)} \leftarrow 0 \;.$$

The eligibility traces can be written as:

$$\vec{\mathbf{e}}_t^{(conj)}(a) \leftarrow \begin{cases} \gamma\lambda_t^{(conj)}\vec{\mathbf{e}}_{t-1}^{(conj)}(a) + \vec{\boldsymbol{\varphi}}(s_t) & \text{if } a = a_t \\ \gamma\lambda_t^{(conj)}\vec{\mathbf{e}}_{t-1}^{(conj)}(a) & \text{otherwise} \end{cases} \tag{55}$$

For episodic tasks γ can be set to 1 (absence). Finally the update rule is identical to (52), where the conjugate eligibility trace is used instead of the fixed λ eligibility trace:

$$\vec{\boldsymbol{\theta}}(a_t) \leftarrow \vec{\boldsymbol{\theta}}(a_t) + \alpha_{ep} \cdot \vec{\mathbf{e}}_t^{(conj)}(a_t) \cdot \delta_t \tag{56}$$

4.7 The policy used to generate actions

A combination of the ε-greedy policy and Gibbs soft-max (Sutton and Barto 1998) policy is used to pick up an action and to strike a balance between exploration and exploitation.

$$\pi_{\varepsilon+Gibbs}(a_i, \vec{\varphi}(s_t)) = Gibbs(a_i, \vec{\varphi}(s_t)) + \Pr(a_i, \vec{\varphi}(s_t)) \tag{57}$$

Using ε-greedy probability allows exploration to be increased as needed by initially setting ε to a high value then decreasing it through episodes.

$$\Pr(a,\vec{\varphi}(s_t)) = \begin{cases} 1-\varepsilon+\dfrac{\varepsilon}{|A|} & if\ \ a = \arg\max\limits_{i}\left[\vec{\varphi}(s_t)^T\cdot\vec{\theta}(a_{(i)})\right] \\ \dfrac{\varepsilon}{|A|} & otherwise \end{cases} \tag{58}$$

The Gibbs soft-max probability given by equation (59) enforces the chances of picking the action with the highest value when the differences between the values of it and the remaining actions are large, i.e. it helps in increasing the chances of picking the action with the highest action-value function when the robot is sure that this value is the right one.

$$Gibbs(a_{(i)},\vec{\varphi}(s_t)) = \frac{\exp\left[\vec{\varphi}(s_t)^T\cdot\vec{\theta}(a_{(i)})\right]}{\sum_{j=1}^{|A|}\exp\left[\vec{\varphi}(s_t)^T\cdot\vec{\theta}(a_{(j)})\right]} \tag{59}$$

4.8 The learning method

The last model component to be discussed is the learning algorithm. The basis of the model learning algorithm is the Sarsa(λ) control algorithm with linear function approximation (Sutton and Barto 1998). However, this algorithm was adapted to use the TD-conj(0) instead of the TD(λ) update rules. Hence, it was denoted as Sarsa-conj(0). From a theoretical point of view, TD-conj(0)– and any algorithm depending on its update such as Sarsa-conj(0) – uses the conjugate gradient direction in conjunction with TD(0) update. While, from an algorithm implementation point of view, according to the equivalency theorem, TD-conj(0) and Sarsa-conj(0) have the same skeleton of TD(λ) and Sarsa(λ) (Sutton and Barto 1998) with the difference that TD-conj(0) and Sarsa-conj(0) use the variable eligibility traces $\lambda_t^{(conj)}$ (Altahhan 2008). The benefit of using TD-conj(0) update is to optimize the learning process (in terms of speed and performance) by optimizing the depth of the credit assignment process according to the conjugate directions, purely through automatically varying λ in each time step instead of assigning a fixed value to λ manually for the duration of the learning process.

Sarsa is an on-policy bootstrapping algorithm that has the properties of (a) being suitable for control, (b) providing function approximation capabilities to deal with huge state space, and (c) applying on-line learning. These three properties are considered ideal for the visual robot homing (VRH) problem. The ultimate goal for VRH is to control the robot to achieve the homing task, the state space is huge because of the visual input, and on-line learning was chosen because of its higher practicality and usability in real world situations than off-line learning.

```
Initialization
  initialize  b, m, and  final_episode
  $n \leftarrow \approx 3 \times \frac{256}{b} \times m$
  $\vec{\boldsymbol{\theta}}_0(a_i) \leftarrow \vec{1} \qquad i = 1:|A|$
  $a_0 \leftarrow 2$
Repeat for each episode
  $\vec{\mathbf{e}}_0(a_i) \leftarrow \vec{\mathbf{0}} \qquad i = 1:|A|$
  $s_0 \leftarrow$ Initial robot view,  $t \leftarrow 1$
  Generate $a_0$ using sampling of probability $\pi(\vec{\boldsymbol{\varphi}}(s_0), a)$
  Repeat (for each step of episode)
    Take action $a_t$ , Observe $r_{t+1}$, $\vec{\boldsymbol{\varphi}}(s_{t+1})$,
    Generate $a_{t+1}$ using sampling of probability $\pi(\vec{\boldsymbol{\varphi}}(s_{t+1}), a)$.
    $\delta_t \leftarrow \left[ r_{t+1} + \gamma\, \vec{\boldsymbol{\varphi}}(s_{t+1})^T \cdot \vec{\boldsymbol{\theta}}(a_{t+1}) - \vec{\boldsymbol{\varphi}}(s_t)^T \cdot \vec{\boldsymbol{\theta}}(a_t) \right]$
    ${}^{1}\lambda_t^{(conj)} \leftarrow \frac{(\delta_t \vec{\boldsymbol{\varphi}}_t - \delta_{t-1} \vec{\boldsymbol{\varphi}}_{t-1})^T \vec{\boldsymbol{\varphi}}_t}{\gamma (\delta_t \vec{\boldsymbol{\varphi}}_t - \delta_{t-1} \vec{\boldsymbol{\varphi}}_{t-1})^T \vec{\mathbf{e}}_{t-1}}$,  ${}^{2}\lambda_t^{(conj)} \leftarrow \frac{\delta_t \|\vec{\boldsymbol{\varphi}}_t\|^2}{\gamma \delta_{t-1} \|\vec{\boldsymbol{\varphi}}_{t-1}\|^2}$, or  ${}^{3}\lambda_t^{(conj)} \leftarrow \frac{(\delta_t \vec{\boldsymbol{\varphi}}_t - \delta_{t-1} \vec{\boldsymbol{\varphi}}_{t-1})^T \vec{\boldsymbol{\varphi}}_t}{\gamma \delta_{t-1} \|\vec{\boldsymbol{\varphi}}_{t-1}\|^2}$
    $\vec{\mathbf{e}}_t^{(conj)}(a) \leftarrow \begin{cases} \gamma \lambda_t^{(conj)} \vec{\mathbf{e}}_{t-1}^{(conj)}(a) + \vec{\boldsymbol{\varphi}}(s_t) & \textit{if } a = a_t \\ \gamma \lambda_t^{(conj)} \vec{\mathbf{e}}_{t-1}^{(conj)}(a) & \textit{otherwise} \end{cases}$
    $\vec{\boldsymbol{\theta}}(a_t) \leftarrow \vec{\boldsymbol{\theta}}(a_t) + \alpha_{ep} \cdot \vec{\mathbf{e}}_t^{(conj)}(a_t) \cdot \delta_t$
    $\vec{\boldsymbol{\varphi}}(s_t) \leftarrow \vec{\boldsymbol{\varphi}}(s_{t+1}),\ \vec{\boldsymbol{\varphi}}(s_{t-1}) \leftarrow \vec{\boldsymbol{\varphi}}(s_t)$
    $\delta_{t-1} \leftarrow \delta_t, \quad a_t \leftarrow a_{t+1}$
  until $\frac{1}{n} \sum_{i=1}^{n} \phi_i(s_t) < \psi_{upper}$
until episode == final_episode
```

Fig. 3. Linear dynamic-policy conjugate gradient Sarsa-conj(0) control, with RBF features extraction, linear action-value function approximation and Policy Improvement. The approximate Q is implicitly a function of $\vec{\theta}$. $\lambda_t^{(conj)}$ can be assigned to any of the three forms calculated in the preceding step.

The action-value function was used to express the policy, i.e. this model uses a critic to induce the policy. Actor-critic algorithms could be used, which have the advantage of simplicity, but the disadvantage of high variance in the TD error (Konda and Tsitsiklis 2000). This can cause both high fluctuation in the values of the TD error and divergence. This limitation was addressed in this model by carefully designing a suitable scheme to balance exploration and exploitation according to a combination of Gibbs distribution and ε-greedy policy. The Gibbs exponential distribution has some important properties which helped in realizing the convergence. According to (Peters, Vijayakumar et al. 2005) it helps the TD error to lie in accordance with the natural gradient.

In that sense this model is a hybrid model. Like any action-value model it uses the action-value function to induce the policy, but not in a fixed way. It also changes the policy preferences (in each step or episode) towards a more greedy policy like any actor-critic model. So it combines with and varies between action-value and actor-critic models. It is felt that this hybrid structure has its own advantages and disadvantages, and the convergence properties of such algorithms need to be studied further in the future.

TD-conj(0) learns on-line through interaction with software modules that feed it with the robot visual sensors (whether from simulation or from a real robot). The algorithm coded as a controller returns the chosen action to be taken by the robot, and updates its policy through updating its set of parameters used to approximate the action-value function Q. Three linear networks are used to approximate the action-value function for the three actions.

$$\vec{\boldsymbol{\theta}}(a_{(i)}) = (\theta_1^{a(i)},\ldots,\theta_i^{a(i)},\ldots,\theta_n^{a(i)}) \qquad i = 1,..|A|.$$

The current image was passed through an RBF layer, which provides the feature vector $\vec{\boldsymbol{\varphi}}(s_t) = (\phi_1,\ldots,\phi_i,\ldots,\phi_n)$. The robot was left to run through several episodes. After each episode the learning rate was decreased, and the policy was improved further through general policy improvement theorem (GPI). The overall algorithm is shown in Fig. 3.

The learning rate was the same used by (Boyan 1999)

$$\alpha_{ep} = \alpha_0 \cdot \frac{n_0 + 1}{n_0 + episode} \tag{60}$$

This rate starts with the same value as α_0 then is reduced exponentially from episode to episode until the final episode. n_0 is a constant that specifies how quickly α_{ep} is reduced. It should be noted that the policy is changing during the learning phase. The Sarsa algorithm evaluates the same policy that it generates the samples from, i.e. it is an on-policy algorithm. It uses the same assumption of the general policy improvement principle to anticipate that even when the policy is being changed (improved towards a more greedy policy) the process should lead to convergence to optimal policy. It moves all the way from being arbitrarily stochastic to becoming only ε-greedy stochastic.

5 Experimental results

The model was applied using a simulated Khepera (Floreano and Mondada 1998) robot in Webots™ (Michel 2004) simulation software. The real Khepera is a miniature robot, 70 mm in diameter and 30 mm in height, and is provided with 8 infra-red sensors for reactive behaviour, as well as a colour camera extension.

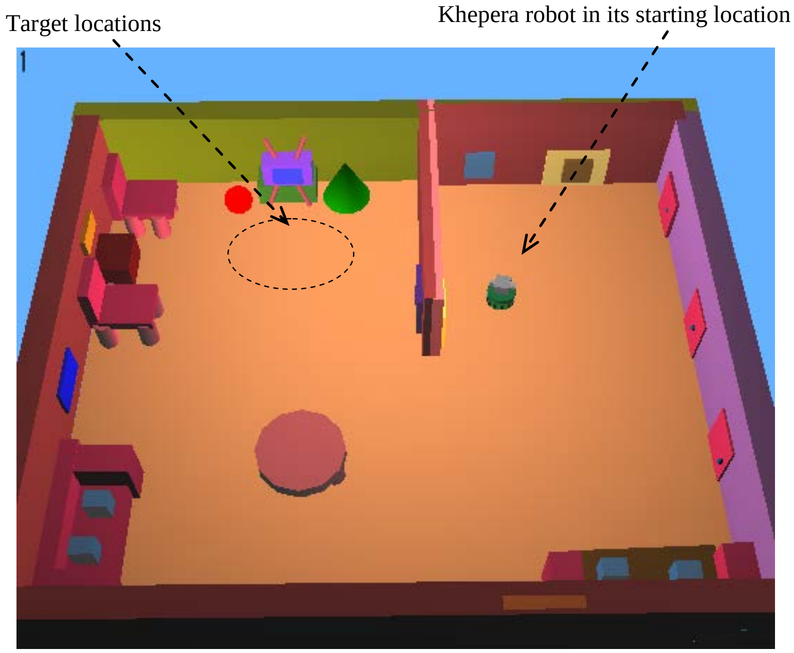

Fig. 4. A snapshot of the realistic simulated environment.

A (1.15 m x 1 m) simulated environment has been used as a test bed for our model. The task is to learn to navigate from any location in the environment to a home location (without using any specific object or landmark). For training, the robot always starts from the same location, where it cannot see the target location, and the end state is the target location. After learning the robot can be placed in any part of the environment and can find the home location.

Fig. 4 shows the environment used. The home is assumed to be in front of the television set. A cone and ball of different colours are included to enrich and add more texture to the home location. It should be re-emphasized that no object recognition techniques were used, only the whole image measure. This allows

the model to be applied to any environment with no constraints and with minimal prior information about the home. The controller was developed using a combination of C++ code and Matlab Engine code.
The robot starts by taking three (m=3) snapshots for the goal location. It then undergoes a specific number (EP) of episodes that are collectively called a run-set or simply a run. In each episode the robot starts from a specific location and is left to navigate until it reaches the home location. The robot starts with a random policy, and should finish a run set with an optimised learned policy.

5.1 Practical settings of the model parameters

Table 1 summarises the various constants and parameters used in the Sarsa-conj(0) algorithm and their values/initial values and updates. Each run lasts for 500 episodes (EP=500), and the findings are averaged over 10 runs to insure validity of the results. The feature space parameters were chosen to be b=3, m=3. Hence, $n = 3 \times (round(256/3)+1) \times 3 = 774$. This middle value for b, which gives a medium feature size (and hence medium learning parameters dimension), together with the relatively small number of stored views (m=3), were chosen mainly to demonstrate and compare the different algorithms on average model settings. However, different setting could have been chosen.
The initial learning rate was set to $\alpha_0 = (1/EP) \times (1/n) \approx (1/500) \times (1/1000) = 2 \times 10^{-6}$ in accordance with the features size and the number of episodes. This is to divide the learning between all features and all episodes to allow for good generalization and stochastic variations. The learning rate was decreased further from one episode to another, equation (60), to facilitate learning and to prevent divergence of the policy parameters $\vec{\theta}$ (Tsitsiklis and Van Roy 1997) (especially due to the fact that the policy itself is changing). Although factor $\lambda_t^{(conj)}$ in TD-conj(0) is a counterpart of fixed λ in conventional TD(λ), in contrast with λ it varies from one step to another to achieve better results. The discount constant was set to $\gamma = 1$, i.e. the rewards sum does not need to be discounted through time because it is bounded, given that the task ends after reaching the final state at time T.

$\psi_{upper}, \psi_{lower}$ are determined empirically and were set to 0.96 and 0.94 respectively when using the NRB measure and b=m=3. These setting simply indicate that to terminate the episode the agent should be ≥ 96% sure (using the NRB similarity measure) that its current view corresponds with one (or more) of the stored views to assume that it has reached the home location. Furthermore, they indicate that the agent should be ≥ 94% sure that its current view is similar to the stored views to assume that it is in the home-at-perspective region.

Symbol	Value	Description
EP	500	Number of episodes in each run
α_0	$\alpha_0 = 2 \times 10^{-6} \approx (1/EP) \times (1/n)$	Initial learning rate
α_{ep}	$\alpha_{ep} = \alpha_0((n_0 \times EP + 1)/\ (n_0 \times EP + ep))$	Episode learning rate
n_0	$75\%EP$	Start episode for decreasing α_{ep}
ε_0	0.5	Initial exploration rate
ε_{ep}	$\varepsilon_{ep} = \varepsilon_0((n_0 \times EP + 1)/\ (n_0 \times EP + ep))$	Episodic exploration rate
n0ε	$50\%EP$	Start episode for decreasing ε_{ep}
γ	1	The reward discount factor
m	3	Number of snapshots of the home
b	3	Features histograms bin size

Table 1. The Model different parameters, their values and their description.

5.2 Convergence results

Fig 5. shows the learning plots for the TD-conj(0)≡ TD(λ_t^{conj}) where the ${}^1\lambda_t^{conj}$ was used. Convergence is evident by the exponential shape of all of the plots. In particular the cumulative rewards converged to an acceptable value. The steps plot resonates with the rewards plot, i.e. the agent attains gradually good performance in terms of cumulative rewards and steps-per-episode. The cumulative changes made to the policy parameters have also a regular exponential shape, which suggests the minimization of required learning from one episode to another. It should be noted that although the learning rate is decreased through episodes, if the model were not converging then more learning could have occurred in later episodes, which would deform the shape of the changes in the policy parameters plot.

λ can take any value in the [0 , 1] interval. It has been shown by (Sutton and Barto 1998) that the best performance for TD(λ) is expected to be when λ has a high value close to 1, such as 0.8 or 0.9 depending on the problem. λ=1 is not a good candidate as it approximates Monte Carlo methods and has noticeably inferior performance than smaller values (Sutton and Barto 1998). It should also be noted that the whole point of the suggested TD-conj(0) method is to optimize and automate the selection of λ in each step to allow TD to perform better and avoid trying different values for λ.

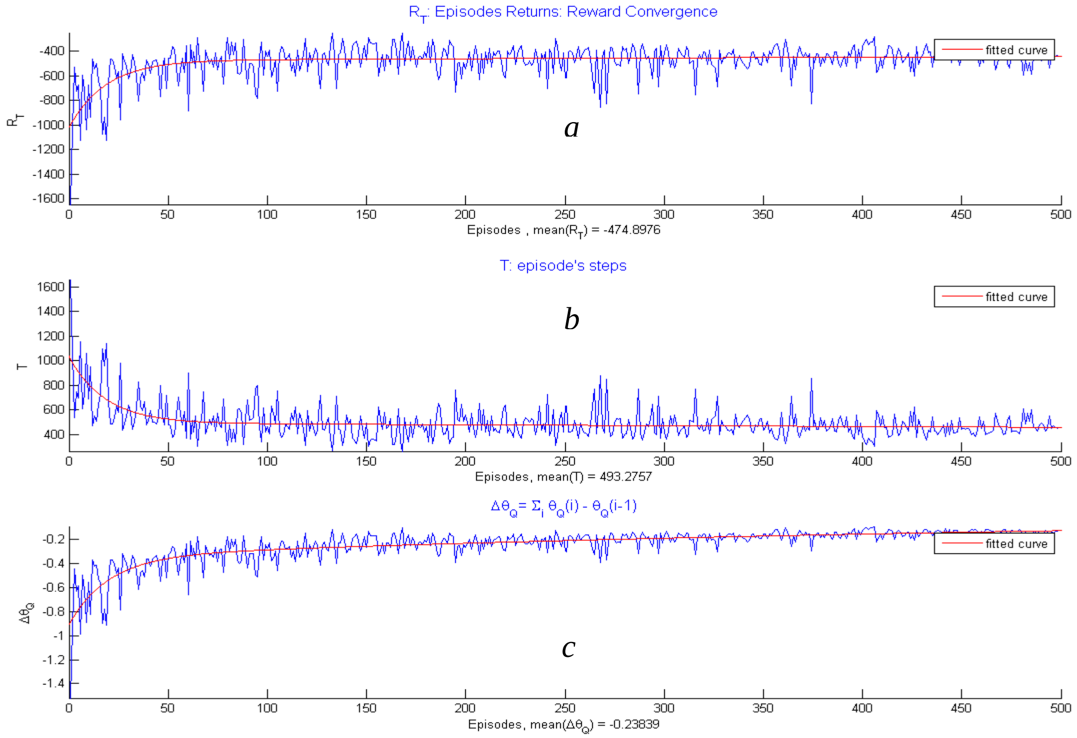

Fig. 5. TD($^1\lambda_t^{(conj)}$) algorithm's performance (a) The cumulative rewards. (b) The number of steps. (c): The cumulative changes of the learning parameters.

Fig. 6. shows that the learning rate decreased per episode. The exploration factor rate was decreased using a similar method. The overall actual exploration versus exploitation percentage is shown in Fig. 6(c). The Temporal error for a sample episode is shown in Fig. 6(d). Fig. 6(e) shows the trace decay factor $^1\lambda_t^{(conj)}$ for the same sample episode. It can be seen that for $^1\lambda_t^{(conj)}$, most of the values are above 0.5. As has been stated in the Equivalency Theorem, there is no guarantee that $\lambda_t^{(conj)}$ satisfies the condition $0 \le \lambda_t^{(conj)} \le 1$. Nevertheless, for most of the values this form of $\lambda_t^{(conj)}$ does satisfy this condition. For those values that did not, it is sufficient to apply the following rule on them:

$$if\,(\lambda_t^{(conj)} > 1) \Rightarrow \lambda_t^{(conj)} \leftarrow 1, \qquad if\,(\lambda_t^{(conj)} < 0) \Rightarrow \lambda_t^{(conj)} \leftarrow 0 \tag{61}$$

It should be noted that better performance could have been achieved by the following rule:

$$if\,(\lambda_t^{(conj)} > 1) \Rightarrow \lambda_t^{(conj)} \leftarrow 1-\xi, \qquad if\,(\lambda_t^{(conj)} < 0) \Rightarrow \lambda_t^{(conj)} \leftarrow 0 \tag{62}$$

However, using this rule would mean that the results shown for TD($\lambda_t^{(conj)}$) might have been affected by the higher performance expected for TD update when $\lambda_t^{(conj)}$ is close (but not equal) to 1. This is because for one single update at some time step t TD(λ) and TD($\lambda_t^{(conj)}$) are identical for the same $\lambda_t^{(conj)}$ value. It is the collective variation from one TD(λ) update to another at each time step that makes TD($\lambda_t^{(conj)}$) different from TD(λ). Therefore, better performance could have been achieved by following rule (62). Hence the performance of TD($\lambda_t^{(conj)}$) could be questionable and shaken when this rule is used. Few values did not satisfy the Equivalency Theorem condition - the percentage was 0.0058% for TD(${}^1\lambda_t^{(conj)}$). To show the path taken by the robot in each episode the Global Positioning System (GPS) was used to register the robot positions but not to aid the homing process whatsoever. Fig. 7. shows the evident improvements that took place during the different learning stages.

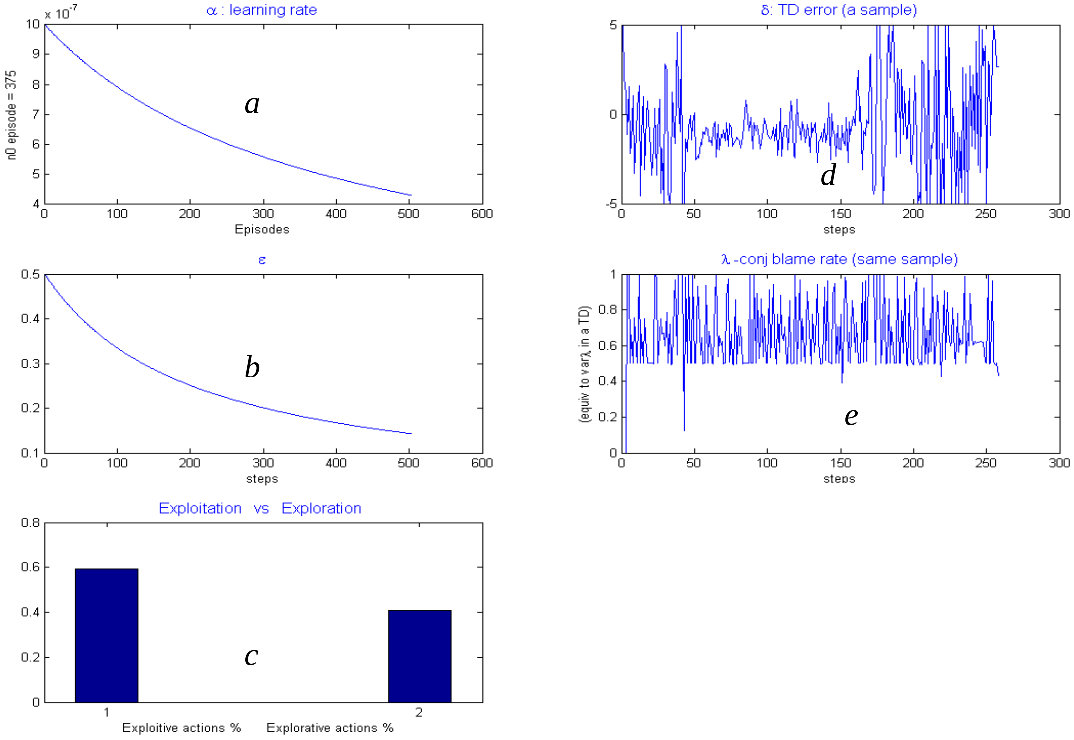

Fig. 6. TD(${}^1\lambda_t^{(conj)}$)internal variables (a): The learning rate. (b): the exploration factor rate. (c): the overall exploration versus exploitation. (d)): the temporal error for a sample episode. (e): the trace decay factor $\lambda_t^{(conj)}$

5.3 Action space and setting exploitation versus exploration

Since action space is finite, and to avoid fluctuation and overshoot in the robot behaviour, low wheel speeds were adopted for these actions. This in turn required setting the exploration to a relatively high rate (almost 50%) during the early episodes. It was then dropped gradually through episodes, in order to make sure that most of the potential paths were sufficiently visited. Setting exploration high also helps to decrease the number of episodes needed before reaching an acceptable performance. This explains the exponential appearance of the different learning curves.

The features variance also played a role in the exploration/exploitation rates. This was because $\hat{\sigma}^2{}_0$ was initialized in the first episode with $\sigma^2(im)$, the variance of the goal location snapshots, then it was updated in subsequent episodes until it was stable. This encourages the agent to explore the environment more in the first episode than any other, which results in big differences between the first and the rest of the episodes. Therefore, it should be noted that all of the episodic figures have been plotted excluding the first episode, to prevent the graphs from being unnecessarily mis-scaled.

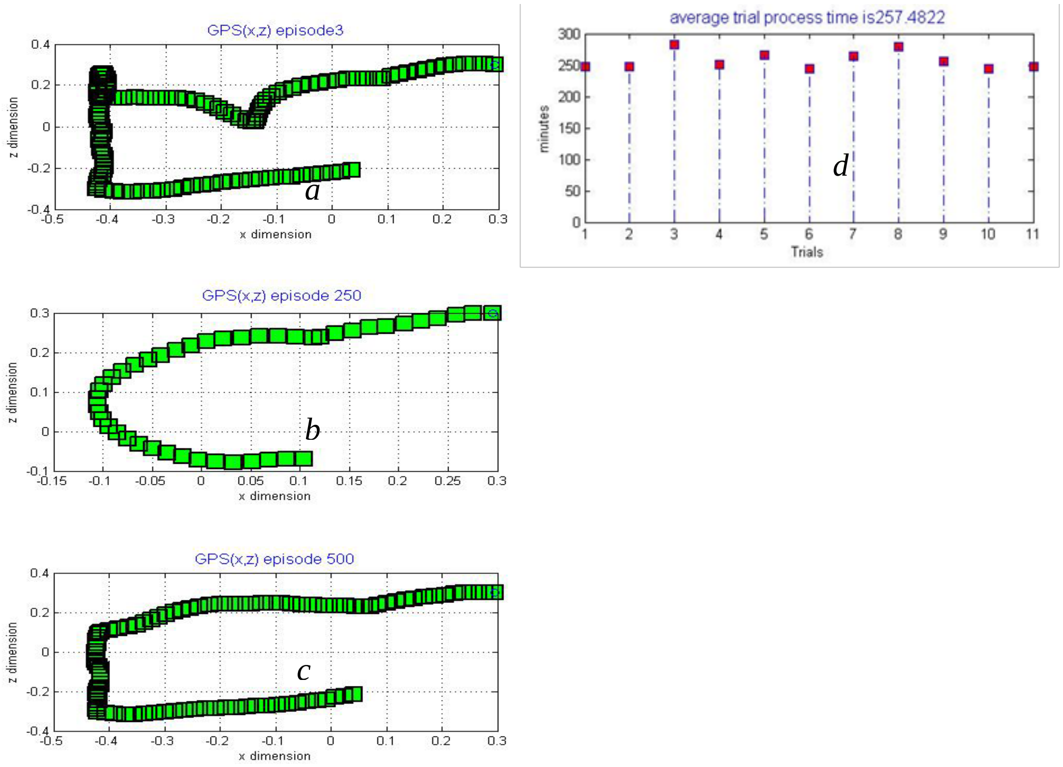

Fig. 7. TD($^{1}\lambda_t^{(conj)}$) algorithm performance for the homing problem (a, b, c): GPS plots for early, middle and last episodes, they show the trajectory improvement that took place during learning. (d): the timing plot of the 10 run sets (trials).

6 TD($\lambda_t^{(conj)}$) and TD(λ) comparisons study

6.1 Rewards comparison

Fig. 8. shows the fitted curves for the rewards plots of the different TD(λ) and TD($\lambda_t^{(conj)}$) algorithms. It summarizes the experiments conducted on the model and its various algorithms in terms of the gained rewards. The model uses Sarsa(λ), and Sarsa($\lambda_t^{(conj)}$), algorithms with a dynamic-policy. It should be recalled that the only difference between the two algorithms is that one uses TD(λ) update and the other uses TD($\lambda_t^{(conj)}$) update. Therefore, the comparisons highlight the differences between TD(λ) and TD-conj(0) updates, and the collective study highlights the dynamic-policy algorithm behaviour.

Several interesting points can be realized in this figure: The three TD($\lambda_t^{(conj)}$) algorithms climb quicker and earlier (50-150 episodes) than the five TD(λ) algorithms then: TD($^{1}\lambda_t^{(conj)}$) and TD($^{2}\lambda_t^{(conj)}$) keeps a steady performance to finally dominate the rest of the algorithms.

While, although the fastest (during the first 150 episodes), TD($^{3}\lambda_t^{(conj)}$) deteriorates after that. TD(λ) algorithms performances varied but they were slower and in general performed worse than TD-conj(0) algorithms: TD(0.4) climbed quickly and its performances declined slightly at the late episodes (after 350 episodes). TD(0) did the same but was slower. TD(0.95, 0.8) climbed slowly but kept a steady improvement until they came exactly under TD($^{1}\lambda_t^{(conj)}$) and TD($^{2}\lambda_t^{(conj)}$). TD(0.9) performed slightly better than TD-conj(0) at a middle stage (150-350), then it declined after that. TD(0.9) was similar to TD($^{3}\lambda_t^{(conj)}$) although it was slower at the rise and the fall.

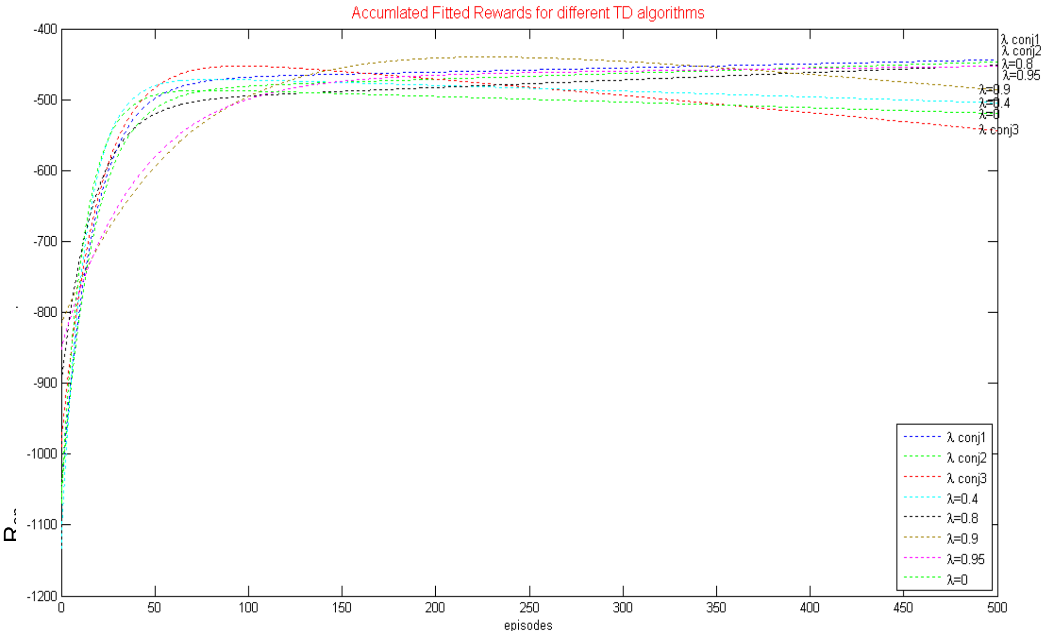

Fig. 8. Overall comparisons of different TD(λ) methods for λ = 0, 0.4, 0.8 and TD($\lambda_t^{(conj)}$) for ${}^1\lambda_t^{(conj)}$ ${}^2\lambda_t^{(conj)}$ and ${}^3\lambda_t^{(conj)}$ using the fitted returns curves for 10 runs each with 500 episodes.

Therefore, for the fastest but short term performance TD(${}^3\lambda_t^{(conj)}$) is a good candidate. For a slower and good performance on the middle run TD(0.9) might still be a good choice. For a slower and long term performance TD(0.8) and TD(0.95) is a good choice.

For both the fastest and best overall performance in the long term TD(${}^1\lambda_t^{(conj)}$) and TD(${}^2\lambda_t^{(conj)}$) are the best choices. Nevertheless, those findings are guidelines and even for the tackled problem they do not tell the whole story. As will be shown in the following section, other performance measure can further give a better picture about those algorithms.

6.2 Comparisons beyond the rewards

¶(6pt)

Fig. 9. summarizes different comparisons between the different algorithms using averages of different performance measures for the same run sets mentioned in the previous section.

¶(14pt)

6.2.1 The figure structure

The set of algorithms that uses TD(λ) updates is shown in blue dots (connected), while the set of the new proposed TD-conj(0) is shown in red squares (not connected). Each measure has been demonstrated in an independent plot.

The horizontal axes were chosen to be the λ value for TD(λ), while for TD-conj(0) there is no specific value for λ as it is varying from one step to another (therefore disconnected). However, for the purpose of comparing TD(${}^1\lambda_t^{(conj)}$) , TD(${}^2\lambda_t^{(conj)}$) and TD(${}^3\lambda_t^{(conj)}$) algorithms, their measures were chosen to be located above (correlate with) TD(0.8), TD(0.9) and TD(0.95) respectively. The arrow to the left of each plot refers to the direction of good performance.

The figure is divided horizontally into two collections; one that is concerned with the during-learning measures and contains six plots distributed along two rows and three columns. The other collection is concerned with the after-learning measures and contains three plots that are distributed over one row.

The first row of the during-learning collection contains measures that are RL related such as the rewards and the parameters changes. The second row of the first collection is associated with the problem under consideration, i.e. the homing task. For testing purposes, the optimal route of the homing task can be designated in the studied environment. The difference between the current position and the desired position can be used to calculate the root mean squared error (RMS) of each time step, which are then summed to form the RMS for each episode, and those in turn can be summed to form the run set RMS. Two different performance measures were proposed using the GPS; their results are included in the second row of the figure. The first depends on the angular difference between the current orientation and the desired orientation and is called the angular RMS. The second depends on the difference between the current step movement vector and the desired movement vector, and is called vectorial RMS (Altahhan 2008). All of these measures was taken during learning (performing 500 episodes) hence the name. All of the during-learning measures in this figure were averaged over 10 run sets each with 500 episodes.

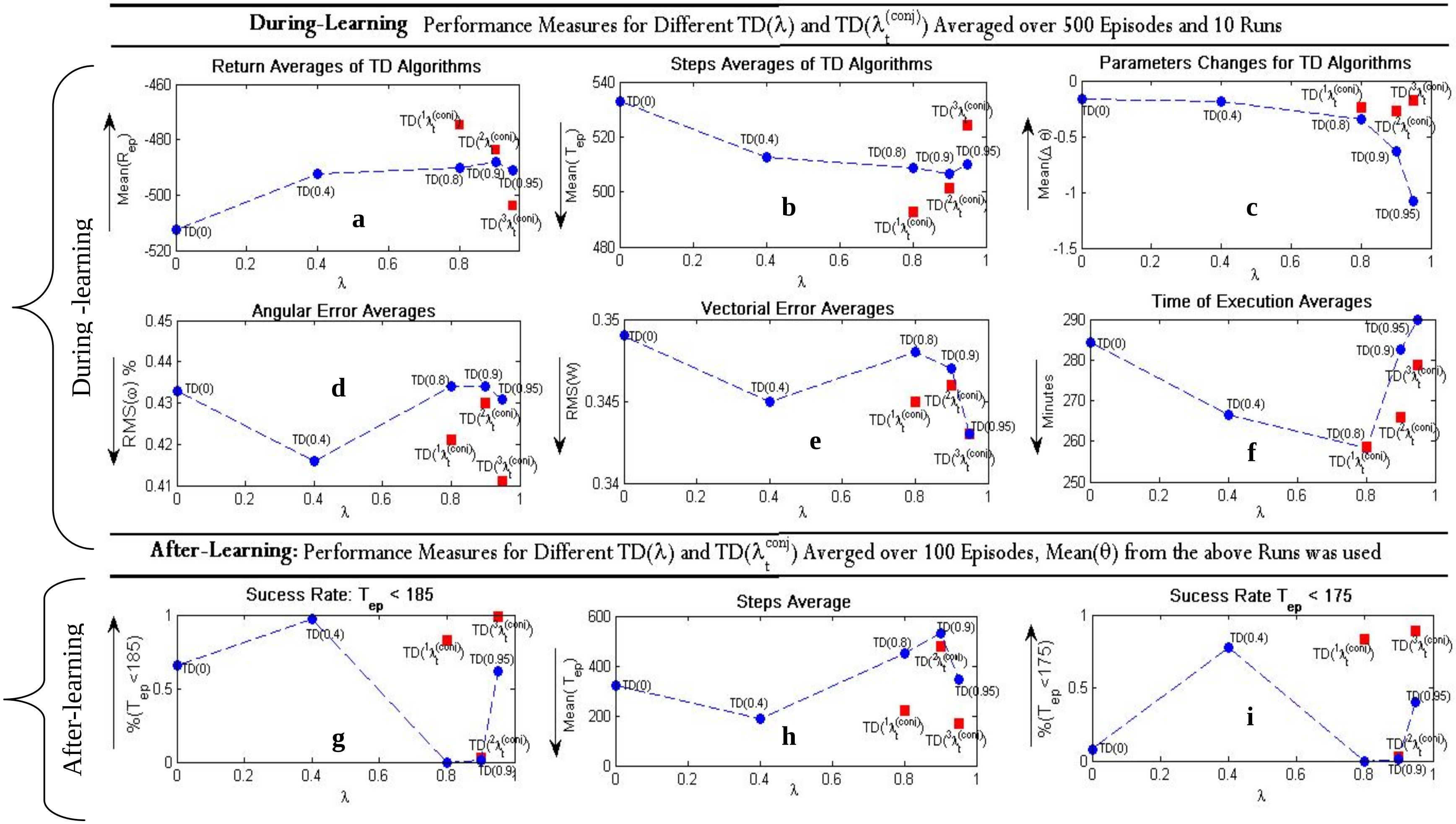

Fig. 9. Overall comparisons of different TD(λ) methods for λ = 0, 0.4, 0.8, 0.9, 0.95 and TD($\lambda_t^{(conj)}$) for ${}^1\lambda_t^{(conj)}$, ${}^2\lambda_t^{(conj)}$ and ${}^3\lambda_t^{(conj)}$ based on Table 6.2; each row is presented as a plot. The arrows refer to the direction of better performance.

The second collection are measures that have been taken after the agent finished the 500 (learning episodes) × 10 (run sets); where it was left to go through 100 episodes without any learning. In those episodes the two policy components (Gibbs and ε-greedy) were restricted to an ε-greedy component only and the policy parameters used are the averaged parameters of all of the previous 10 run sets (performed during-learning). The Gibbs component was added initially to allow the agent to explore the second best guess more often than the third guess (action). Nevertheless, keeping this component after learning would increase the arbitrariness and exploration of the policy which is not desired anymore therefore it was removed.

The three measures in the after-learning collection are related to the number of steps needed to complete the task. The steps average measures the average number of steps needed by the agent to complete the task. Another two scales measures the rate of successes of achieving the task within a pre-specified number of steps (175 and 185 steps[1]).

¶(14pt)

6.2.2 The algorithms assessment

To assess the performance of the algorithms TD(λ) algorithms will be analyzed first then the focus is moved to TD-conj(0). Apparently, when the blue dots are examined in the first row, it can be seen that (a and b) appears to have a theme for TD(λ); the number of steps and the accumulated rewards both have a soft peak. The best TD(λ) algorithm is TD(0.9). This suggests that the peak of TD(λ) for the particular studied homing task is near that value. During-learning TD(0.9) could collect the most rewards in the least number of steps, however it caused more changes than the rest (except for TD(0.95)).

When the red squares are examined in the first row of plots it can be seen that TD(${}^{1}\lambda_t^{(conj)}$) and TD(${}^{2}\lambda_t^{(conj)}$) performed best in terms of the gained rewards (as was already pointed out in the previous subsection). What is more, they incurred less changes than any other algorithm which is an important advantage over other algorithms.

6.2.3 Calculations efficiency and depth of the blame

There are some interesting points to note when examining the changes in the policy parameters. Mean Δθ appears to have a theme for TD(λ); the changes to θ increase with λ. When the Mean Δθ is compared for the best two algorithms (during learning) TD(${}^{1}\lambda_t^{(conj)}$) and TD(λ=0.9), it can be seen that TD(${}^{1}\lambda_t^{(conj)}$) caused less changes to the learning parameters but still outperformed TD(λ=0.9). TD(${}^{1}\lambda_t^{(conj)}$) avoids the unnecessary changes for the policy parameters and hence avoids fluctuations of performance during learning. It only performed the necessary changes. On the other hand TD(λ=0.9) always 'blames' the previous states trace equally for all steps (because λ is fixed) and maximally (because λ=0.9 has a high value). TD(${}^{1}\lambda_t^{(conj)}$) gets the right balance between deep and shallow blame (credit) assignment by varying the deepness of the trace of states to be blamed and incurs changes according to the conjugate gradient of the TD error.

6.2.4 Time efficiency

The execution time Mean(Time) provides even more information than Mean Δθ. Both TD(${}^{1}\lambda_t^{(conj)}$) and TD(λ=0.9) have almost identical execution times, although the execution time for TD(${}^{1}\lambda_t^{(conj)}$) was initially anticipated to be more than any TD(λ) because of the extra time for calculating $\lambda_t^{(conj)}$. This means that with no extra time cost or overhead TD(${}^{1}\lambda_t^{(conj)}$) achieved the best performance, which are considered to be important results.

TD(${}^{2}\lambda_t^{(conj)}$) performed next best, after TD(${}^{1}\lambda_t^{(conj)}$), according to the Mean(R_T) and Mean(T) performance measures, but not in terms of policy parameters changes or execution time; for those, TD(λ=0.9) still

[1] These two numbers were around the least number of steps that could be realized by the agent without highly restricting its path, they were empirically specified.

performed better. This means that this form of ${}^{2}\lambda_t^{(conj)}$ achieved better performance than λ=0.9, but at the cost of extra changes to the policy parameters, and incurred extra time overhead for doing so.

6.2.5 Quickest learner and RMS

When the RMS errors in the second row are examined TD(${}^{3}\lambda_t^{(conj)}$) comes to the scene as the best TD-conj(0) algorithm as it has the least errors. Further, when examining the after-learning collection it can be seen that TD(${}^{3}\lambda_t^{(conj)}$) also performed best. This comes surprising as it should be TD(${}^{1}\lambda_t^{(conj)}$) TD(${}^{2}\lambda_t^{(conj)}$) who are expected to get all the acclaim. On the other hand when the RMS for TD(λ) are examined it can be realized that in terms of the problem errors TD(0.4) performed best since its errors are the least of all the studied TD(λ) algorithms. Further, when examining the after-learning collection it can be seen that TD(0.4) also performed best. This comes equally surprising as it should be TD(0.9) who is expected to get all the praise.

The reason for both situations can be deduced from Fig. 8. the fastest two algorithms in terms of gaining the most rewards in the early episodes are TD(0.4) and TD(${}^{3}\lambda_t^{(conj)}$). Both algorithms got the best results after-learning because even though their performances have deteriorated after the first 150 episodes of the during-learning, however significant amount of learning takes place during this early stage because the learning rate is reduced quickly according to equation (59). It can be seen that TD(0.4) (the light green line) was the fastest out of all the TD(λ) algorithms and that TD(${}^{3}\lambda_t^{(conj)}$) was also the fastest in terms of gaining the most rewards in the early stages. Both scored the best in the after-learning because they learned very quickly and because most of the learning takes place in the early stages in the proposed model. Still why these algorithms have deteriorated after that stage? This is because exploration was higher than these algorithms could keep up with. Compared to the intricate learning that these algorithms achieved in a short time the high exploration caused a noisy learning which in turn deteriorated the algorithms performance in the later episodes without very much affecting what they have already gained in the early episodes. This can be added to the dis/advantages of those algorithms. So the mid range decay traces are more subtle to explorative action which might distort their learning. As for who is better TD(0.4) or TD(${}^{3}\lambda_t^{(conj)}$), it is apparent form the angular error RMS(ω) and the after learning measures that TD(${}^{3}\lambda_t^{(conj)}$) is the best.

6.2.6 ε-Greedy divergence

ε-greedy divergence is a divergence that occurs after learning when the agent changes from the decreasing ε-greedy-Gibbs policy to a fixed ε-greedy policy. It has occurred sometimes especially when the agent had to switch from the reinforcement learning behaviour to the reactive behaviour near the walls and obstacles. For example the TD(λ=0.9) and TD(${}^{2}\lambda_t^{(conj)}$) diverged in this way. Also using the walls more is the likely cause that made the RMS(w) of TD(0.95) to beat the RMS(w) of TD(0.4).

6.2.7. Recapitulation

So Who Wins? It can be construed that TD(${}^{1}\lambda_t^{(conj)}$) outperformed all of the described algorithms during learning, while TD(${}^{3}\lambda_t^{(conj)}$) outperformed all of the described algorithms after learning. TD(${}^{1}\lambda_t^{(conj)}$) and TD(${}^{2}\lambda_t^{(conj)}$) suite more a gradual learning process while TD(${}^{3}\lambda_t^{(conj)}$) suits quick and more aggressive learning process. TD(${}^{1}\lambda_t^{(conj)}$) might still be preferred over the other updates because it preformed collectively best in all of the proposed measures (during and after learning). This demonstrates that using the historically oldest form of conjugate factor β to calculate ${}^{1}\lambda_t^{(conj)}$, proposed by Hestenes and Steifel, has performed the best of the three proposed TD(λ_t^{conj}) algorithms. The likely reason is that this form of $\lambda_t^{(conj)}$ uses the preceding eligibility trace in its denominator, equation (40), not only the current and previous gradients.

The TD-conj methods has the important advantage over TD(λ) of automatically setting the learning variable $\lambda_t^{(conj)}$ equivalent to λ in TD(λ), without the need to manually try different λ values. This is the most important contribution of this new method which has been verified by the experimental comparisons. TD-conj gives a canonical way of automatically setting λ to the value which yields the best overall performance.

7. Conclusions and future work

In this chapter a new family of RL methods, TD-conj(λ), has been established by using the conjugate gradient direction instead of the gradient direction in the conventional TD(λ). More importantly, TD-conj(0) was proved to be equivalent to TD(λ_t^{conj}) when both are used with function approximation. λ_t^{conj} is a variable λ that establishes a way of automatically deepening or shallowing the blame trace in each time step to maximize the algorithm's performance. It is hoped that this finding would be of benefit to boost future/existing algorithms that use TD(λ) update because it requires minimal additional implementation and experiments suggest that it costs little or no overhead. A comprehensive comparison study was conducted on both families of algorithms through various measures that uncover the strengths and weaknesses of them for a homing task.

TD(λ_t^{conj}) and the conventional TD(λ) families of algorithms were utilized to learn a homing task in the context of a novel robot visual homing model. The proposed homing model has a straightforward and effective learning procedure without a need to build a map for the environment explicitly. No human intervention is required, no pre- or manual processing is required, and no a priori knowledge about the environment is needed (landmarks etc), with the added advantage of solving the robot abduction problem instantly. The only required information is in the form of *m* stored views of the home that could be fed to the robot independently of the learning process.

A number of enhancements are planned for the future. Although setting up a suitable exploration/exploitation was automated in the model and required only the specification of ε_0 and n_{ε_0} prior to execution, finding the best balance between these parameters needs more examination. Also increasing/decreasing the robot speed proportional to the increase/decrease in the reward is another enhancement that is worth looking at. Finally, reducing the number of the learning parameters by reducing the sparsity of the feature space (feature selection) is another issue that is being investigated.

8. References

Altahhan, A. (2008). Conjugate Gradient Temporal Difference Learning for Visual Robot Homing. Faculty of Computing and Engineering. Sunderland, University of Sunderland. **Ph.D.:** 206.

Altahhan, A., K. Burn, et al. (2008). Visual Robot Homing using Sarsa(λ), Whole Image Measure, and Radial Basis Function. International Joint Conference on Neural Networks (IJCNN), Hong Kong.

Anderson, A. M. (1977). "A model for landmark learning in the honey-bee." Journal of Comparative Physiology A **114**: 335-355.

Argyros, A. A., K. E. Bekris, et al. (2001). Robot Homing based on Corner Tracking in a Sequence of Panoramic Images. 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01)

Arkin, R. C. (1998). Behavior-Based Robotics, MIT Press.

Asadpour, M. and R. Siegwart (2004). "Compact Q-learning optimized for micro-robots with processing and memory constraints." Robotics and Autonomous Systems, Science Direct, Elsevier.

Baird, L. C. (1995). Residual Algorithms: Reinforcement Learning with Function Approximation. International Conference on Machine Learning, proceedings of the Twelfth International Conference, San Francisco, CA, Morgan Kaufman Publishers.

Bhatnagar, S., R. S. Sutton, et al. (2007). Incremental Natural Actor-Critic Algorithms. Neural Information Processing Systems (NIPS19).

Boyan, J. A. (1999). Least-squares temporal difference learning Proceedings of the Sixteenth International Conference on Machine Learning San Francisco, CA Morgan Kaufmann.

Cartwright, B. A. and T. S. Collett (1987). "Landmark maps for honeybees." Biological Cybernetics **57**: 85-93.

Falas, T. and A.-G. Stafylopatis (2001). Temporal differences learning with the conjugate gradient algorithm. Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on, Washington, DC, USA.

Falas, T. and A.-G. Stafylopatis (2002). Temporal differences learning with the scaled conjugate gradient algorithm. Neural Information Processing ICONIP 2002.

Floreano, D. and F. Mondada (1998). Hardware solutions for evolutionary robotics. First European Workshop on Evolutionary Robotics, Berlin, Springer-Verlag.

Gillner, S., A. M. Weiß, et al. (2008). "Visual homing in the absence of feature-based landmark information." Cognition **109**(1): 105-122.

Hagan, M. T., H. B. Demuth, et al. (1996). Neural Network Design, PWS Publishing Company.

Kaelbling, L. P., M. L. Littman, et al. (1998). "Planning and acting in partially observable stochastic domains." Artificial Intelligence **101**: 99–134.

Konda, V. and J. Tsitsiklis (2000). Actor-critic algorithms. . NIPS 12.

Lazaric, A., M. Restelli, et al. (2007). Reinforcement Learning in Continuous Action Spaces through Sequential Monte Carlo Methods. NIPPS 2007.

Michel, O. (2004). "Webots: Professional Mobile Robot Simulation." International Journal of Advanced Robotic Systems **1**: 39-42.

Murphy, R. R. (2000). Introduction to AI Robotics. Cambridge, Massachusetts., The MIT Press.

Muse, D., C. Weber, et al. (2006). "Robot docking based on omnidirectional vision and reinforcement learning." Knowledge-Based Systems, Science Direct, Elsevier **19**(5): 324-332

Nehmzow, U. (2000). Mobile robotics: A Practical Introduction, Springer-Verlag.

Nocedal, J. and S. J. Wright (2006). Numerical Optimization. New York, Springer.

Peters, J., S. Vijayakumar, et al. (2005). "Natural Actor-Critic." Proceedings of the Sixteenth European Conference on Machine Learning: 280–291.

Rubner, Y. and et al. (2000). "The Earth Mover's Distance as a Metric for Image Retrieval." International Journal of Computer Vision **40**(2): 99-121.

Schoknecht, R. and A. Merke (2003). "TD(0) Converges Provably Faster than the Residual Gradient Algorithm." Machine Learning **20**(2): 680-687.

Sheynikhovich, D., R. Chavarriaga, et al. (2005). Spatial Representation and Navigation in a Bio-inspired Robot. Biomimetic Neural Learning for Intelligent Robots. S. Wermter, M. Elshaw and G. Palm, Springer**:** 245-265.

Simmons, R. and S. Koenig (1995). Probabilistic Robot Navigation in Partially Observable Environments. Proc .of the International Joint Conference on Artificial Intelligence. .

Stone, P., R. S. Sutton, et al. (2005). "Reinforcement learning for robocup soccer keepaway." International Society for Adaptive Behavior **13**(3): 165–188.

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

Sutton, R. S. and A. Barto (1998). Reinforcement Learning, an introduction. Cambridge, Massachusetts, MIT Press.

Szenher, M. (2005). Visual Homing with Learned Goal Distance Information. Proceedings of the 3rd International Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2005), Awara-Spa, Fukui, Japan, Springer.

Thrun, S. (2000.). Probabilistic Algorithms in Robotics, CMU-CS-00-126. .

Thrun, S., W. Burgard, et al. (2005). Probabilistic Robotics. Cambridge, Massachusetts; London, England, The MIT Press.

Thrun, S., Y. Liu, et al. (2004). "Simultaneous localization and mapping with sparse extended information filters." International Journal of Robotics Research **23**(7-8): 693–716.

Tomatis, N., I. Nourbakhsh, et al. (2001). Combining Topological and Metric: a Natural Integration for Simultaneous Localization and Map Building. Proc. Of the Fourth European Workshop on Advanced Mobile Robots (Eurobot 2001).

Tsitsiklis, J. N. and B. Van Roy (1996). "Feature-based methods for large scale dynamic programming. ." Machine Learning **22**: 59-49.

Tsitsiklis, J. N. and B. Van Roy (1997). "An analysis of temporal-difference learning with function approximation." IEEE Transactions on Automatic Control **42**(5): 674–690.

Ulrich, I. and I. Nourbakhsh (2000). Appearance-Based Place Recognition for Topological Localization IEEE International Conference on Robotics and Automation San Francisco, CA.

Vardy, A. (2006). Long-Range Visual Homing. IEEE International Conference on Robotics and Biomimetics, 2006. ROBIO '06., Kunming.

Vardy, A. and R. Moller (2005). "Biologically plausible visual homing methods based on optical flow techniques." Connection Science **17**(1–2): 47–89.

Vardy, A. and F. Oppacher (2005). A scale invariant local image descriptor for visual homing. Biomimetic neural learning for intelligent robots. . G. Palm and S. Wermter, Springer.

Weber, C., D. Muse, et al. (2006). "A camera-direction dependent visual-motor coordinate transformation for a visually guided neural robot." Knowledge-Based Systems, Science Direct, Elsevier **19**: 348–355.

Weber, C., S. Wermter, et al. (2004). "Robot docking with neural vision and reinforcement." Knowledge-Based Systems, Science Direct, Elsevier **17**(2-4): 165-172.

Weber, K., S. Venkatesh, et al. (1999). "Insect-inspired robotic homing." Adaptive Behavior **7**(1): 65-97.

Zhang, C., S. Abdallah, et al. (2008). Efficient multi-agent reinforcement learning through automated supervision. International Conference on Autonomous Agents Estoril, Portugal.

Ziv, O. and N. Shimkin (2005). Multigrid Methods for Policy Evaluation and Reinforcement Learning. IEEE International Symposium on Intelligent Control, Limassol.