2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

A Fast Learning Variable Lambda TD Model

Used to Realize Home Aware Robot Navigation

Abdulrahman Altahhan (*Author*)
College of Computer Information Technology
American University in the Emirates
United Arab Emirates
abed.tahhan@gmail.com

***Abstract*— This work describes a fast learning robot goal-aware navigation model that employs both gradient and conjugate gradient Temporal Difference (TD, TD-conj) methods. It builds on the fact that TD-conj was proven to be equivalent to a gradient TD method with a variable lambda under certain conditions. Based on straightforward features extraction process combined with goal-aware capabilities provided by whole image measure, the model solves what we call u-turn-homing benchmark problem without using landmarks. Only one goal snapshot was used with agent facing the goal directly. Therefore a novel threshold stopping formula was used to recognize the goal which is less sensitive to the agent-goal orientation problem. Unlike other models, this model refrains from artificially manipulating or assuming a priori knowledge about the environment, two constraints that widely restrict the applicability of existing models in realistic scenarios. An on-line control method was used to train a set of neural networks. With the aid of variable and fixed eligibility traces, these networks approximate the agent's action-value function allowing it to take close to optimal actions to reach its home. The effectiveness of the model was experimentally verified on an agent.**

Keywords—TD-conj; Home Aware; Variable λ TD; U-Turn-Homin; Orientation Insensitive Thersholding

I. INTRODUCTION

Reinforcement learning (RL) with function approximation has been shown in some cases to converge slowly [1]. Bootstrapping methods like temporal difference (TD) [2] although was proved to be faster than other RL methods, such as residual gradient established by Baird [3], it can still be slow [4]. TD can be speed up by using it with other gradient types. In [1], for example, TD along with the natural gradient has been used to boost learning.

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. Actor-critic, for example, uses TD update in its critic part. It maintains a structure for choosing the actions according to a learned policy (the actor), that is separate from the structure to estimate the value-function (the critic). In those methods, when the gradient of the performance function is used to explicitly estimate the policy (policy gradient methods), convergence was shown to be slow because of the high variance of their gradient estimates [1]. Other RL methods such as Q-learning, although convenient for control, can suffer from both divergence and slow convergence [5].

Numerical techniques have been used with RL methods to speed up its performance. For example, [6] 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, [7], by using a supervised approach combined with RL to enhance the model performance.

The idea of using the conjugate gradient (CG) with TD has been explored before [8, 9]. Their early experiments confirmed that using such a combination can enhance the performance of TD. In their study [10] a special emphasis is given on the implementation details where an effort is made to save computation costs by adopting the Møller's scaled conjugate gradient algorithm (SCG) [11]. SCG differs from CG in that it avoids the line search for the step size α_t by introducing a new factor ρ that is raised or lowered within each iteration, during execution, in order to regulate the indefiniteness of the Hessian matrix of the error function. 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 the former gap.

The outline of the paper is as follows. Firstly, a variable λ TD is summarized. Then a detail description of the novel navigation model and its components is presented in section 3. The extensive simulation experiments and their results are shown in sections 4 and 5, followed by conclusion and further work in section 6.

II. VARIABLE LAMBDA TD

A. Biological Conformation and Plausibility

There can be many ways to vary λ, and some might be heuristic [1]. However, there is an interesting canonical way

that is underpinned by following the conjugate gradient of the TD error instead of the gradient [12]. This forces the eligibility traces to vary the deepness of the blame (credit assignment), for taking previous decisions, depending on the current TD error that is constantly opposed according to the conjugate gradient direction.

The performance advantage of using this type of variable λ TD over fixed λ TD is confirmed by the comparative study performed by [12]. In summary, those advantages are recapitulated by: acquiring better or comparable performance with less training and parameters changes and with less execution time. In other word, the conjugate variable λ TD is more efficient than fixed λ TD. What is more, using variable λ TD has a biological justification of varying the level of learning readiness (attentiveness) from a situation to another, depending on previous experience and other internal factors, e.g. attention. Also it can be related to varying the deepness of thinking, where it is not always beneficial to propagate deeply through past experience in every coming situation; some situations need shallow thinking for the best interest of the animal.

Results from [12] confirms that fixing λ to force the learning process to take advantage of the longest possible past experience is not always the right approach. It is argued that doing so can exhaust the organ without apparent advantages. Whereas, varying the deepness of the eligibility traces, although might initially be thought to have inferior performance, has been proved to achieve a comparable, if not better, results than forcing the agent to think always to the deepest possible extent.

We think that this is an interesting and important proposition and it worth to bring the attention to it by presenting a model that employs it successfully. This paper presents a comprehensive biologically-inspired model for robot navigation that uses variable λ TD to achieve autonomy in learning to visually navigate towards a pre-specified goal location. To the best knowledge of the author variable λ TD has not been used before in robot navigation by other researchers.

B. TD-Conj Methods

In general TD is a reinforcement learning method that tries to calculate or approximate the expected future accumulated rewards/cost due to following some policy π. A policy is a mapping between a states and actions that specifies what the agent should do in a certain state s. It guides this process through the TD error δ_t which is defined as:

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

Where value function $V_t(s_t)$ is the expected future accumulated rewards, s_t denotes a state at time step t, r is for the reward/cost and γ is a decay factor. $V_t(s_t)$ can be approximated through a linear approximation of parameters $\vec{\theta}_t$ and state's features $\vec{\phi}_t$ as:

$$V_t(s_t) = \vec{\theta}_t^T \vec{\phi}_t \tag{2}$$

$$\nabla_{\vec{\theta}_t} V_t(s_t) = \vec{\phi}_t \tag{3}$$

In this case an update rule for fixed λ TD can be written as:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t \delta_t \cdot \vec{e}_t \tag{3}$$

$$\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \vec{\phi}_t \tag{4}$$

Here $\vec{e}_t$ is the eligibility trace that specifies how deep the blame is to be for the current cost. While the update rules for conjugate variable λ TD can be written as:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha_t \delta_t \vec{e}_t^{(conj)} \tag{5}$$

$$\vec{e}_t^{(conj)} = \gamma\lambda_t^{(conj)} \vec{e}_{t-1}^{(conj)} + \vec{\phi}_t \tag{6}$$

Where $\lambda_t^{(conj)}$ can be given in any of the following forms:

$$^1\lambda_t^{(conj)} = \frac{\left(\delta_t\vec{\phi}_t - \delta_{t-1}\vec{\phi}_{t-1}\right)^T\vec{\phi}_t}{\gamma\left(\delta_t\vec{\phi}_t - \delta_{t-1}\vec{\phi}_{t-1}\right)^T\vec{e}_{t-1}},\ ^2\lambda_t^{(conj)} = \frac{\delta_t}{\gamma\delta_{t-1}}\cdot\frac{\left\|\vec{\phi}_t\right\|^2}{\left\|\vec{\phi}_{t-1}\right\|^2},\ ^3\lambda_t^{(conj)} = \frac{\left(\delta_t\vec{\phi}_t - \delta_{t-1}\vec{\phi}_{t-1}\right)^T\vec{\phi}_t}{\gamma\delta_{t-1}\left\|\vec{\phi}_{t-1}\right\|^2} \tag{7}$$

Since $\gamma \in [0,1]$ then $\lambda_t^{(conj)}$ should satisfy that $\lambda_t^{(conj)} \in [0,1]$. From an application point of view, it suffices to reset $\lambda_t^{(conj)}$ value to the boundaries of this condition whenever its value goes beyond those boundaries[1].

C. Biological Interpretation of $\lambda_t^{(conj)}$

It can be realized that both δ_t and δ_{t-1} are involved in the calculations of eligibility traces. This means that the relative rate-of-changes between consequent steps of TD error plays an important role in changing $\lambda_t^{(conj)}$. For simplicity we discuss the second form of $\lambda_t^{(conj)}$ which involves two terms; the first is the rate of change of TD error δ_t / δ_{t-1} and the second is the rate of change of the norm of current and previous feature vectors $\left\|\vec{\phi}_t\right\|^2 / \left\|\vec{\phi}_{t-1}\right\|^2$. Both has a biological interpretation and $\lambda_t^{(conj)}$ tries to strike a good balance between both. The δ_t / δ_{t-1} term implies that if we compared the current and previous errors and found that the previous error is bigger it means that we should allow future steps to blame the decision made at the previous step. While, if we found that the current error is bigger than the previous error it means that we need to allow future steps to blame the current step mainly without propagating the error to previous steps as they are guiltless of the current error. Cognitively, it means either that "I must have done a mistake in past states so in the future I will blame those estimates", or "it seems that I am doing some mistakes in the current state so in the future I will blame this state estimate". Also the second term is more problem specific. It expresses that, if there are more features present in the current state than the previous,

[1] Any method that depends on TD updates such as Sarsa or Q-learning can take advantage of the more efficient and more biologically-plausible update rules of conjugate variable λ TD

then this state is more important and it should undergo more learning in the future than the previous state, and vice versa.

III. 3. NAVIGATION VARIABLE Λ SARSA MODEL

For visual navigation 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 [3]. In turn, parameterization permits learning a value function representation that can easily accommodate 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 should strike 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.

In a previous work done by the author three snapshots has been taken by the agent for the goal in order to accommodate for the different angles in which the robot can come towards the goal which worked well. However, this has expanded the features vectors and contributed to a more ambiguous ways of specifying the similarity threshold through trial and error. This is because the other two angle snapshots were acting as dissimulators for the goal when the agent is coming from the third angle and has made the best positions features less similar to the goal. It was thought that such processing is redundant and unnecessary if a better approach can be found which should allow the agent to use one snapshot for the goal. The idea is to compare the current image with one direct snapshot only, but at the same time find a parameterized specification for the similarity threshold. Here we are referring to the threshold that specify whether the current image is similar enough with the goal to issue a stopping command for the agent declaring that it reaches the goal and achieved the current episodic learning to move to the next episode.

A. 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. This effectively encodes the input state space into a coarser feature space. Therefore, for a coloured image, the histogram of each colour channel is a vector of components; each component contains the number of pixels that lie in the component's interval (bin).

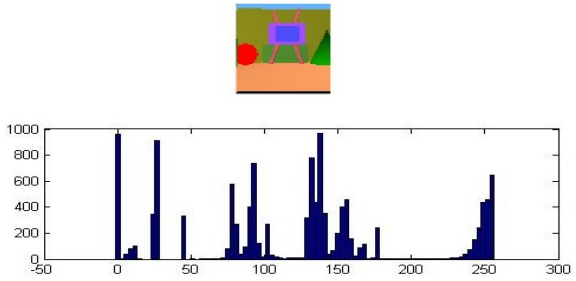

Fig. 1. Sample of a stored view taken for a goal location with its associated histograms in a simulated environment.

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 [4].

The histogram alone 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 recognition of the home from different angles must be accommodated either by different snapshots from different angles or by the similarity measure itself that is able to tell whether a goal was reached form an angle. In this work the latter approach has been taken. One snapshot of the home location is taken before the learning stage. This snapshot defines the goal location and is the only information required to allow the agent to learn to reach its goal 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). Figure 1 shows a sample of a stored view of a goal location and its associated histogram taken in a simulated environment.

B. Features vectors and differential radial basis representation

The difference between a histogram of each channel of the current view and those of the stored views is taken and passed through a radial basis function (RBF) component. Hence it is called differential radial basis representation. This provides the features space $\Phi : S \to \Re^n$ representation (9) which is used with the Sarsa algorithm, described later:

$$\phi_i\left(s_t(c,j)\right) = \exp\left(-\frac{\left(h_i\left(s_t(c)\right) - h_i\left(v(c,j)\right)\right)^2}{2\hat{\sigma}_i^2}\right) \qquad (8)$$

Index t stands for the time step, j for the j[th] 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 of the jth

stored view, $h_i(v(c,j))$ is histogram bin i of channel $v(c,j)$, and $h_i(s_t(c))$ is histogram bin i of channel c of the current view. The number of bins will have an effect on the structure and richness of this representation and hence on the model.

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

$$\hat{\sigma}_i^2 = (1/T-1)\sum_{t=1}^{T}\Delta h_i^2(t) \tag{9}$$

$$\Delta h_i^2(t) = \left(h_i(s_t(c)) - h_i(v(c,j))\right)^2 \tag{10}$$

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 where we have that:

$$\sum_i^n h_i(v(c,j)) = \sum_i^n h_i(s_t(c)) = H \tag{11}$$

assuming that n is the number of features. It can be realized that H is a constant and is equal to the number of all pixels taken for a view. Consequently, the final feature calculation takes the form:

$$\phi_i(s_t(c,j)) = \exp\left(-\frac{\left(h_i(s_t(c)) - h_i(v(c,j))\right)^2}{2H^2\hat{\sigma}^2}\right) \tag{12}$$

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 three channels and each stored view, as follows:

$$\Phi(s_t) = \bigoplus_{j=1}^{m}\bigoplus_{c=1}^{3}\bigoplus_{i=1}^{B}\phi_i(s_t(c,j)) = (\phi_1,\ldots,\phi_i,\ldots,\phi_n) \tag{13}$$

where m is the number of stored views for the goal location, and B is the number of bins to be considered. A value in the range of [0 255] will be used for each pixel, hence the dimension of the feature space is given by:

$$n = 3\times B\times m = 3\times(round(\frac{256}{b})+1)\times m \tag{14}$$

Where b is the bin's size. 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.

C. 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:

$$NRB(s_t) = \sum_{i=1}^{n}\bar{\phi}_i(s_t)\Big/n \tag{15}$$

D. Action space and cyclic learning

In order to avoid the complexity of dealing with a set of actions each with infinite 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]. 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.

On the other hand, an episode describes the collective (action, reward, state) learning steps an agent needs to navigate from any starting location in the environment until it reaches the goal location. The agent is assumed to finish an episode and be in the goal location (final state) if its similarity measure indicates with high certainty that its current view is similar to one of the stored views. This specifies the episode termination condition of the model. *If* $NRB(s_t) \geq \psi_{upper}$ $\Rightarrow$ 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 goal-at-perspective.

E. Setting the reward/cost signal

Each time step the robot spend without finding the goal location has a cost of -1/t. The position signal, expressed by the current similarity NRB_t, is added to the reward/cost signal. Thus, as the current location differs less from the home location, the reward will increase. The third signal is the increase/decrease in similarity between the current step and the previous step, which we call the approaching reward. This signal is defined as $NRB_t - NRB_{t-1}$. Hence, the reward can be rewritten in the following form:

$$r = -1/t + 2NRB_t - NRB_{t-1} \tag{16}$$

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.

F. Variable action eligibility traces and TD update rules

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 [6]. 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.

$$\vec{\mathbf{e}}_t^{(conj)}(a) \leftarrow \begin{cases} \gamma\lambda_t^{(conj)}\vec{\mathbf{e}}_{t-1}^{(conj)}(a) + \vec{\boldsymbol{\varphi}}(s_t) & \text{if } a = a_t \\ \gamma\lambda_t^{(conj)}\vec{\mathbf{e}}_{t-1}^{(conj)}(a) & \text{otherwise} \end{cases} \quad (17)$$

$\lambda_t^{(conj)}$ is conjugate variable discount rate for eligibility traces $\vec{\mathbf{e}}_{t-1}^{(conj)}$ of action a. It is given in (8). 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 action eligibility trace and the episodically changed learning rate α is as follows:

$$\vec{\boldsymbol{\theta}}(a_t) \leftarrow \vec{\boldsymbol{\theta}}(a_t) + \alpha \cdot \vec{\mathbf{e}}_t(a_t) \cdot \delta_t \quad (18)$$

G. The learning algorithm

Figure 2 shows the model learning algorithm. The basis of the model learning algorithm is the Sarsa(λ) control algorithm with linear function approximation [6]. However, this algorithm was changed to use the TD($\lambda_t^{(conj)}$) instead of the TD(λ) update rules. Hence, it was denoted as Sarsa($\lambda_t^{(conj)}$). The benefit of using TD($\lambda_t^{(conj)}$) 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. $\lambda_t^{(conj)}$ can be calculated using any of the three forms of (8) although the algorithm shows ${}^2\lambda_t^{(conj)}$.

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 robot visual navigation problem. (a) The ultimate goal for solving this problem is to control the robot to reach the goal location using vision sensor, (b) the state space is huge because of the visual input, and (c) on-line learning is preferred because of its higher practicality and usability in real world situations than off-line learning.

An action-value function $Q(s,a)$, is similar to the value function $V(s)$ but is confined to a certain action. It calculates the expected future accumulated rewards stemming from: applying action a when agent in state s then following some policy π. It is primarily used to take advantage of the policy improvement theorem. This theorem states that the policy can be improved by increasing the probability of selecting the action with the highest $Q(s,a)$.

The algorithm learns on-line through interaction with software modules that feed it with the robot visual sensors. 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 for the three actions. $\vec{\boldsymbol{\theta}}(a_{(i)}) = (\theta_1^{a(i)},\ldots,\theta_i^{a(i)},\ldots,\theta_n^{a(i)}) \quad i = 1,..|A|$.

The current image was passed through an RBF layer, which provides the feature vector $\vec{\boldsymbol{\varphi}}(s_t) = (\phi_1,\cdots\phi_i,\cdots\phi_n)$. The robot was left to run through several episodes. After each episode the learning rate was decreased, and the policy was improved further.

Initialization

initialize b, m, and final_episode

$n \leftarrow\approx 3 \times \frac{256}{b} \times m, \quad \vec{\boldsymbol{\theta}}_0(a_i) \leftarrow \vec{1} \quad i = 1\cdots|A|, \quad a_0 \leftarrow 2$

$mult \leftarrow 12$

$\psi_{upper} = 1 - \left(\pi\sqrt{mult}\right)^{-1}, \psi_{lower} = 1 - \left(\pi\sqrt{mult-1}\right)^{-1}$

Repeat for each episode

$\vec{\mathbf{e}}_0(a_i) \leftarrow \vec{\mathbf{0}} \quad i = 1\cdots|A|$

$s_0 \leftarrow$ Initial robot view , $t \leftarrow 1$

Generate a_0 using sampling of probabilit y $\Pi(\vec{\boldsymbol{\varphi}}(s_0), a, \varepsilon_0)$

Repeat (for each step of episode)

Take action a_t , Observe r_{t+1}, $\vec{\boldsymbol{\varphi}}(s_{t+1})$,

$a_{best} = \arg\max_{a_i}(\vec{\boldsymbol{\varphi}}(s_t)^T \cdot \vec{\boldsymbol{\theta}}(a_i)), \quad i = 1\cdots|A|$

Generate a_{t+1} using sampling of probabilit y $\Pi(a_{i=1\cdots3}, \varepsilon, \tau)$.

$\delta_t \leftarrow \left[r_{t+1} + \gamma\,\vec{\boldsymbol{\varphi}}(s_{t+1})^T \cdot \vec{\boldsymbol{\theta}}(a_{t+1}) - \vec{\boldsymbol{\varphi}}(s_t)^T \cdot \vec{\boldsymbol{\theta}}(a_t)\right]$

$\lambda_t^{(conj)} \leftarrow \frac{\delta_t \|\vec{\boldsymbol{\varphi}}_t\|^2}{\gamma\delta_{t-1}\|\vec{\boldsymbol{\varphi}}_{t-1}\|^2}$

$\vec{\mathbf{e}}_t^{(conj)}(a) \leftarrow \begin{cases} \gamma\lambda_t^{(conj)}\vec{\mathbf{e}}_{t-1}^{(conj)}(a_i) + \vec{\boldsymbol{\varphi}}(s_t) & \text{if } a_i = a_t \\ \gamma\lambda_t^{(conj)}\vec{\mathbf{e}}_{t-1}^{(conj)}(a_i) & \text{otherwise} \end{cases}, \; i = 1\cdots3$

$\vec{\boldsymbol{\theta}}(a_t) \leftarrow \vec{\boldsymbol{\theta}}(a_t) + \alpha \cdot \vec{\mathbf{e}}_t^{(conj)}(a_t) \cdot \delta_t$

$\vec{\boldsymbol{\varphi}}(s_t) \leftarrow \vec{\boldsymbol{\varphi}}(s_{t+1}), \; \vec{\boldsymbol{\varphi}}(s_{t-1}) \leftarrow \vec{\boldsymbol{\varphi}}(s_t)$

$\delta_{t-1} \leftarrow \delta_t, \quad a_t \leftarrow a_{t+1}$

until $\frac{1}{n}\sum_{i=1}^{n}\phi_i(s_t) \geq \psi_{upper}$

$decreas\_rate \leftarrow (ep_0 + 1)/(ep_0 + episode)$

$\varepsilon \leftarrow \varepsilon_0 \cdot decreas\_rate, \quad \alpha \leftarrow \alpha_0 \cdot decreas\_rate$

until *episode == final_episode*

Fig. 2. Algorithm of conjugate variable λ Sarsa control, with RBF features extraction, linear action-value function approximation and dynamic exploration policy.

H. Stopping Thresholds and von Mises Distribution

Directional or circular statistics provide a comprehensive framework that deals with the orientation data. It deals with directions and rotation in multidimensional spaces. Probabilities that are defined to deal with lines or variable that can increase to infinity is not suitable to deal with data that repeat itself intervaly such as robot orientation. Images taken in this way varies in a behaviour that is not suitable to be described by normal statistics, it must be addressed through

directional statistics. von Mises distribution can be used as an approximation for a warped normal distribution for the normal distribution. Gauss defined the standard normal as having variance $\sigma^2 = 1/2$. However other alternative definitions exist, for example Stephen Stigler defined the standard normal distribution with variance $\sigma^2 = 1/(2\pi)$, in this case the standard distribution will be defined as $f(x) = e^{-\pi x^2}$. This definition is more appealing as it simplify the function itself of course on the account of making the variance more complicated, one can also define $\sigma^2 = 1/(2\pi)^2$. Similarly, wrapped normal distribution can be defined through different variances if it is desired to be localized further around its mean. We used a multiplier of 1/π to specify the variance which in turn specifies the boundaries of the confidence interval for our similarity measure which is the mean of the features. We will construct our threshold by assuming that we are fitting a uniformed warped normal distribution around the maximum similarity measure NRB_{max}=1, which can be assigned a variance σ = 1/multiples of π to recognise its rotational behaviour. Hence, one can construct the following confidence interval $[1-(m\pi)^{-0.5}, 1+(m\pi)^{-0.5}]$. Of course half of this interval is impossible to achieve as it exceeds the maximum similarity, however the first half is where we are interested which specifies the upper threshold $[1-(m\pi)^{-0.5}, 1]$. Another thing to realize is that if the multiple is assumed to be the square root of another integer then one can specify an easy way to vary the interval with slow increments, which is suitable for the problem in hand, hence another empirical interval is $[1-\pi^{-1}m^{-0.5}, 1]$. Figure 3 show the change rate of both ways of specifying the threshold. It should be noted that the second is more desirable as it will make the difference between ψ_{upper} and ψ_{lower} easily achieved by decrementing the multiplier by 1. Equation (19) gives a rule of setting the stopping threshold.

$$\psi_{upper} = 1-\left(\pi\sqrt{mult}\right)^{-1}, \quad \psi_{lower} = 1-\left(\pi\sqrt{mult-1}\right)^{-1} \tag{19}$$

We chose multipliers 11 and 12 (through trial and error) that yields the values 0.904 0.908 respectively to represents the max and min similarities (ψ_{upper}, ψ_{lower}) that specify the termination conditions for our robot. It should be noted that changing the multiplier is much easier to tune the model instead of changing continues values such as ψ_{upper}, ψ_{lower}.

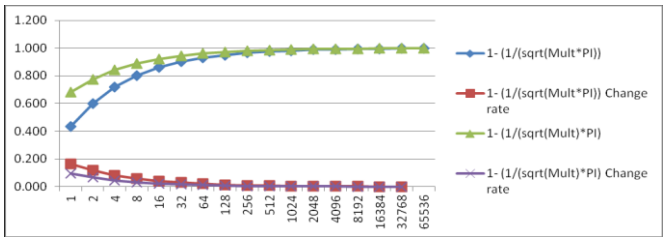

Fig. 3. Change rate of different settings for the Threshold ψ_{upper}

I. *Dynamic exploration policy*

The action-value function was used to express the policy, and control the robot accordingly. The policy model has two components: An exponential component that takes the action-value as input. This component has been scaled first to avoid the problem of raising large number to a power to get a softmax effect.

$$Scaled(a_i) = \frac{\vec{\varphi}(s_t)^T \cdot \vec{\theta}(a_i)}{\sum_j^{|A|} \vec{\varphi}(s_t)^T \cdot \vec{\theta}(a_j)} \tag{20}$$
$$SoftMax(a_i, \tau) = e^{Scaled(a_i, \vec{\varphi}(s_t))/\tau}$$

The second component is a greedy scheme also participated in the action selection process:

$$\varepsilon Greedy(a_i, \varepsilon) = \begin{cases} 1-\varepsilon+\varepsilon/|A| & if \;\; a_i == a_{best} \\ \varepsilon/|A| & otherwize \end{cases} \tag{21}$$

$$\Pi(a_i, \varepsilon, \tau) = \frac{SoftMax(a_i, \tau) + \varepsilon Greedy(a_i, \varepsilon)}{\sum_j^{|A|} SoftMax(a_j, \tau) + \varepsilon Greedy(a_j, \varepsilon)} \tag{22}$$

An important thing to note here is that ε is passed as an argument for the ε-greedy component. i.e. ε has been varied during learning. Early episode had a large ε to encourage the agent to explore its environment more, while late episode had less exploration tendency. Same applies for τ for early and late episodes. This makes our policy a dynamic exploration policy, something to that distinguishes our model. It should be noted that convergence for such dynamic policies has not been confirmed before, and this paper dose not study this convergence form theoretical point of view, instead it show that such dynamic policy model can still converges to an optimal policy. The Gibbs exponential distribution has some important properties which helped in realizing the convergence. According to [7] it helps the TD error to lie in accordance with the natural gradient. Both ε (exploration rate) and α (learning rate) have been exponentially reduced form one episode to another according to the following decrease rate [8]:

$$decreas\_rate = (ep_0 + 1)/(ep_0 + episode) \tag{23}$$

Where ep_0 is the initial episode that specifies how quickly to decrease both entities.

IV. Experiments and Model Settings

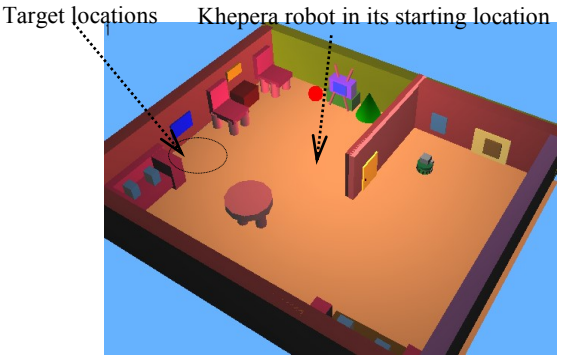

Fig. 4. A snapshot of the realistic simulated environment.

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.

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. Figure 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=1) 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.

A. Practical settings of the model parameters

Table 1 summarises the various constants and parameters used in the Sarsa($^{2}\lambda_t^{(conj)}$) algorithm and their values/initial values and updates. Each run lasts for 20 episodes (EP=20), and the findings are averaged over 5 runs to insure validity of the results. The feature space parameters were chosen to be b=3, m=1.

TABLE I. THE MODEL PARAMETERS, THEIR VALUES AND THEIR DESCRIPTION

Symbol	***Value***	***Description***
EP	20	Number of episodes in each run
α_0	$\alpha_0 = (10^{-1})/(2\times EP^2) = 1.25\times10^{-3}$	Initial learning rate
ε_0	$0.3\times EP$	Initial exploration rate
ep_0	$0.3\times EP$	Start episode for decreasing α and
γ	1	The reward discount factor
m	1	Number of snapshots of the home
b	3	Features histograms bin size
ψ_{upper}, ψ_{lowe}	0.904 0.908	Goal_at_perspective thresholds

Hence, $n = 3\times(round(256/3)+1) = 258$ features. the value for b, which gives a medium feature size (and hence medium learning parameters dimension), together with the minimum number of stored views (m=1), were chosen mainly to the algorithms performance on average model settings. However, different setting could have been chosen. The initial learning rate was set to $\alpha_0 = (10^{-1})/(2\times EP^2) = 1.25\times10^{-3}$ in accordance with the number of episodes. This is to divide the learning between all episodes to allow for good generalization and stochastic variations. The learning rate was decreased further from one episode to another, to facilitate learning and to prevent divergence of the policy parameters $\vec{\theta}$ [9] (especially due to the fact that the policy exploration rate is changing). 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.

V. RESULTS

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 navigation process whatsoever. Figure 5. (c and d) shows the evident improvements that took place during the different learning episodes. Evidently, good performance was started to occur in early episodes (episode 9) but it was not always sustained because the agent had to keep looking for better policy, especially in early episodes. GPS gives a problem-specific assessment about the performance of the algorithm and have strong indication that the algorithms is working well for the problem (navigation).

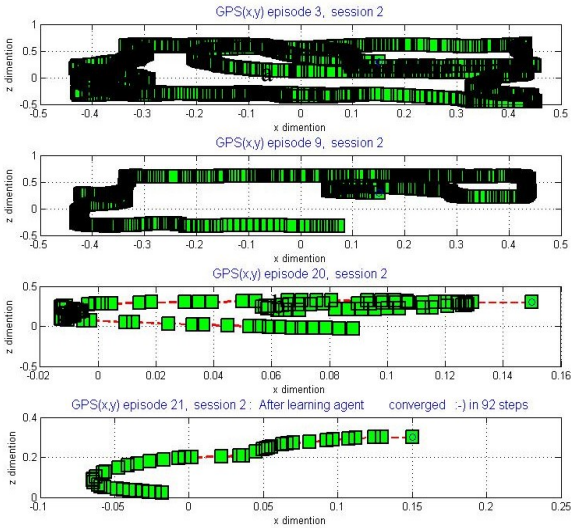

Fig. 5. TD(conj) algorithm's performance performance using GPS, it can be seen that after learning the agent made a u-turn very efficently

Figure 6. shows the learning plots for the TD($^{1}\lambda_t^{(conj)}$). Convergence is evident by the decreased number of steps needed in each episode. 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 would have occurred in later

episodes, which would deform the shape of the changes in the policy parameters plot.

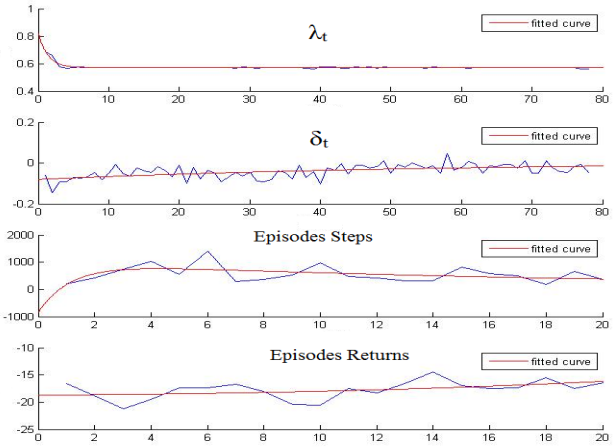

Fig. 6. TD-conj algorithm performance for the homing problem

A. So why the model is faster?

The better performance of this model can be attributed to the following enhancements. First Less number of initial snapshots, in fact only one snapshot has been taken for the goal from one angle (facing the goal). Better Termination condition as shown in previous section. Better calibration of Gibbs for the exploration/exploitation rate. Better and simpler reactive behavior. Learning is continued during reactive behaviour. Cost is changed to 1/t (t is time step) to penalize early steps more than latter steps as normally those will have more profound effect on the future than late steps for this benchmark problem. Finally, high penalty was set for wall hitting. Wall hitting is included in the learning process (in previous work [12] it was paused until the reactive behavior is finished) now reactive takes control but the neural network will be charged penalty during that time as long as the robot is hitting walls. This proved to be moderately effective in preventing costal behavior or blind walking where the robot hit two or three walls before reaching the target.

VI. CONCLUSIONS

This work presents a robot navigation model that employs a form of conjugate variable λ TD and a homing technique to realize full autonomous navigation. It utilizes a novel thresholding technique that uses von Mises distribution to reduce sensitivity to the goal orientation. Comparing to previous work this model achieved convergence in a small number of episodes (20) in comparison with previous model which had to go through hundreds of episodes. This makes this model appealing for industrial and home realistic application. We bring the attention to the capability of the proposed model which is not confined to homing in the conventional sense where the home is on sight. It goes beyond homing to navigating towards a hidden goal location autonomously. We would like also to point out that, after learning, robot learns to navigate towards the goal location starting from almost any location in the environment.

The model learns to tackle the u-turn problem to reach a hidden goal without human intervention, 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 three views of the goal location that the robot itself takes and stores automatically before starting the learning process.

Therefore the proposed approach for learning to navigate towards a hidden-goal is extremely practical and portable, and depends entirely on automatic learning. If it can find its way to industry all what the operator would need to do is to show the robot its goal location then sit and watch it learning to reach the goal from anywhere in the environment. This approach is also interesting for the design of multi navigational tasks model where a robot can learn to reach each goal independently using a dedicated linear neural network. Then some prioritization technique can be used to switch between those inexpensive neural networks. This potential is what we intend to investigate in the future.

References

[1] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Incremental Natural Actor-Critic Algorithms," presented at Neural Information Processing Systems (NIPS19), 2007.

[2] R. S. Sutton, "Learning to predict by the methods of temporal differences," Machine Learning, vol. 3, pp. 9–44, 1988.

[3] L. C. Baird, "Residual Algorithms: Reinforcement Learning with Function Approximation," presented at International Conference on Machine Learning, proceedings of the Twelfth International Conference, San Francisco, CA, 1995.

[4] R. Schoknecht and A. Merke, "TD(0) Converges Provably Faster than the Residual Gradient Algorithm," Machine Learning, vol. 20, pp. 680-687, 2003.

[5] V. Konda and J. Tsitsiklis, "Actor-critic algorithms. ," presented at NIPS 12, 2000.

[6] O. Ziv and N. Shimkin, "Multigrid Methods for Policy Evaluation and Reinforcement Learning," presented at IEEE International Symposium on Intelligent Control, Limassol, 2005.

[7] C. Zhang, S. Abdallah, and V. Lesser, "Efficient multi-agent reinforcement learning through automated supervision," presented at International Conference on Autonomous Agents Estoril, Portugal, 2008.

[8] T. Falas and A.-G. Stafylopatis, "Temporal differences learning with the conjugate gradient algorithm," presented at Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on, Washington, DC, USA, 2001.

[9] T. Falas and A.-G. Stafylopatis, "Temporal differences learning with the scaled conjugate gradient algorithm," presented at Neural Information Processing ICONIP 2002, 2002.

[10] T. Falas and A.-G. Stafylopatis, "Implementing Temporal-Difference Learning with the Scaled Conjugate Gradient Algorithm," Neural Processing Letters, vol. 22, pp. 361 - 375, 2005.

[11] M. F. Møller, "A scaled conjugate gradient algorithm for supervised learning," Neural Networks, vol. 6, pp. 525-533, 1993.

[12] A. Altahhan, "A Robot Visual Homing Model that Traverses Conjugate Gradient TD to a Variable λ TD and Uses Radial Basis Features," in Advances in Reinforcement Learning, A. Mellouk, Ed. Vienna: InTech Education and Publishing, 2011, pp. 225-254.