Visual Robot Homing using Sarsa(λ), Whole Image Measure, and Radial Basis Function.

Abdulrahman Altahhan, Kevin Burn, Stefan Wermter

Hybrid Intelligent Systems Research Group,
School of Computing and Technology,
University of Sunderland, SR6 0DD UK www.his.sunderland.ac.uk

abdulrahman.altahhan@sunderland.ac.uk.

Abstract—**This paper describes a model for visual homing. It uses Sarsa(λ) as its learning algorithm, combined with the Jeffery Divergence Measure (JDM) as a way of terminating the task and augmenting the reward signal. The visual features are taken to be the histograms difference of the current view and the stored views of the goal location, taken for all RGB channels. A radial basis function layer acts on those histograms to provide input for the linear function approximator. An on-policy on-line Sarsa(λ) method was used to train three linear neural networks one for each action to approximate the action-value function with the aid of eligibility traces. The resultant networks are trained to perform visual robot homing, where they achieved good results in finding a goal location. This work demonstrates that visual homing based on reinforcement learning and radial basis function has a high potential for learning local navigation tasks.**

I. Introduction

A skill which plays an integral role in achieving robot autonomy is the ability to learn to operate in *a priori* unknown environments[1]. *Visual homing* is the act of finding a goal location by comparing the image currently viewed with stored 'snapshot' images (normally taken while animal or robot is heading off its home location). *Visual navigation* is the act of navigating form one location to the other in the environment, as efficiently as possible. In this paper we present a model for visual homing, which can also be used in local navigation, using reinforcement learning (RL from now on) and an online snapshot comparison technique. This snapshot comparison facilitates online learning and execution in *a priori* unknown environments to reach a goal location[1].

Robotics borrows several concepts from animal homing and navigation strategies described in the biological literature [2, 3]. While both visual homing and visual navigation are related, they have been kept fairly apart due to the fact that visual homing is more inspired by the biology and due to the fact that visual navigation is more general than visual homing. Nevertheless, navigation can be accomplished more directly by using local homing strategies to reach some location, without directly building a map or using a model of environment dynamics. The limitation is that the learned strategies to navigate to home is bound to that particular location. Therfore, if the robot needs to navigate to a different location, it should be trained to do so. We argue that our model can also be used for general navigation tasks due to the fact that it can operate in any environment and requires no additional effort except showing the robot, online or offline, its goal location, then letting it trains.

Algorithms based on the snapshot model [3] propose various strategies for finding features within images and establishing correspondence between them in order to determine a home direction. Block matching, for example, takes a block of pixels from one image and searches for the best matching block in another image within a fixed search radius [4]. The degree of match between blocks is usually judged by the Sum of Squared Differences (SSD) or some other local correlation measure[5]. In our model we will take a more effective approach by comparing bins of histograms through a Radial Bases Function layer, and using images only taken around the home, nothing more.

Reinforcement Learning has been used previously in robotics navigation and control problems. Several of the models that used it are inspired by biological findings, e.g. [6]. Although successful, some of those models lack the generality and/or practicality, and some are restricted to their environment. The model proposed by [7] for example depends heavily on object recognition of a landmark in the environment to achieve the task. We have addressed this issue in our model by avoiding object recognition and using a whole image measure technique instead, to measure the dissimilarity of current and goal views to identify whether the robot reached the goal location (with the desired orientation). This was possible with no prior knowledge or constrains regarding those images. By adding the above advantage to the learning robustness and generality of RL, coupled with *visual* states and rewards, the model achieved a high level of robustness, generality, and applicability.

While 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. Although the immediate execution of model-based

[1] Note: goal location and home location will be used interchangeably in this paper.

navigation system can be successful [8, 9], RL techniques have got the advantages of model-free systems i.e. there is no knowledge needed prior to operating the robot. It learns the best policy for the environment dynamics. While the idea of using snapshots to do robot localization is not new [10], visual homing based on reinforcement learning and radial basis input layer and whole image measure is a novel contribution of this paper.

We begin by presenting an overview of our reinforcement learning context and Markov Decision Processes (MDP) framework followed by the Temporal Difference (TD) learning algorithm for continuous states space. This is followed by a detailed description of our model, demonstrating generality and simplicity of execution. Then we present empirical results of a robot reaching a goal location visually in a simulation environment.

II. Background Of Reinforcement Learning

Reinforcement learning concerns the problem of learning to predict the sum of rewards an agent is receiving while interacting with its environment in order to optimally execute a task [11]. Instead of being given examples of the desired behavior, the learning agent must find out - using its environment feedback and using gradual explorative actions - how to act best to execute a task. Usually this feedback is a minimal signal of reward or punishment induced in some way in the environment. This signal is called the reinforcement signal.

In any environment there exists a set of *states* that represent the situations that the agent can face (or recognize). Those states define the state space denoted by *S*, which can be finite or infinite and continuous. The *actions* are those simple activities the agent is able to do in a certain state. The set of those actions define the *actions space A*. Those actions can also be finite or infinite. The environment normally reacts or responds to any action taken by the agent by returning a signal indicating or reinforcing how good or bad this action was for the task. It is called the *reward* signal or the reinforcement signal. The dynamics of an environment are the set of probability distributions that distinguish its internal properties. Those are mainly the *state transition function* and the *reward function*.

The state transition function is a probability distribution defined on the state space that specifies the probability of moving form state *s* at time *t* to another state *s'* at time *t*+1 after applying action *a*:

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$$

The reward function is defined as the expected reward returned by the environment for each state after applying a certain action:

$$\Re_{ss'}^{a} = E\{r_t | s_t = s, s_{t+1} = s', a_t = a\}$$

where r_t is the actual reward returned by the environment and fully observed by the agent.

As with most reinforcement work, we will restrain ourselves to the Markovian environments. A Markov decision process (MDP) is defined by a tuple $(S, A, P_{ss'}^{a}, R_{ss'}^{a}, \gamma)$, where $\gamma \in [0,1]$ is a discount rate parameter, and where the Markov property is satisfied [9, 11]. A trajectory of experience is a sequence $s_1, a_1, r_2, s_2, a_2, r_3, \ldots$ where the agent in s_1 takes action a_1 then receives reward r_2 and transitioning to s_2 before taking a_2, etc.

A *policy* π specifies (probabilistically or deterministically) the action that needs to be taken for each different state.

$$\pi : S \times A \to [0,1], \qquad \sum_a \pi(s,a) = 1 \cdot$$

where $\pi(s,a)$ is the probability of selecting action *a* when an agent is in state *s*. A deterministic policy is a mapping between states and actions $\pi : S \to A$. The ultimate goal of reinforcement learning methods (algorithms) is to learn an optimum policy that, when followed, maximizes the accumulated rewards *expected* to be gained by the agent during interaction with its environment. This is normally reached through estimating the expected sum in some form since a model of the environment is normally not available and undesired to be a requirement. Even in methods that assume a model of the environment dynamics to be known, such as Dynamic Programming methods, the expectation still needs to be estimated due to the bootstrapping characteristic of such a method. By bootstrapping we mean building on an own initial estimation to reach a better estimation closer to the real value [11].

The discounted sum of rewards at time step t is called the *return* R_t where:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (1)$$

Expected accumulated rewards for a certain policy π can be expressed in two forms: the value function $V^{\pi}(s)$ and the action-value function $Q^{\pi}(s,a)$. A *value function* for a policy is defined as: $V^{\pi}(s) : S \to \Re$.

V^{π} specifies the expected return (sum of rewards r_t) from the starting state *s* and onwards. Obviously each policy has a different value function, hence the upper superscript.

$$V^{\pi}(s) = E_{\pi}\left[R_t | s = s_t\right] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right] \quad (2)$$

The central idea of RL is to try to learn an estimate of the value function of the adopted policy depending on the interaction between the agent and its environment. In other words, to predict the value function of the agent's MDP policy. An essential property of the value function can be deduced from the intrinsic recursion it posses:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right] \\ &= \sum_a \pi(s,a) \sum_s P_{ss'}^{a} \left[\Re_{ss'}^{a} + \gamma V^{\pi}(s')\right] \quad (3) \end{aligned}$$

The action-value function is defined as

$Q^{\pi}(s,a): S \times A \rightarrow \Re$, where:

$$Q^{\pi}(s,a) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s = s_t, a = a_t\right] \quad (4)$$

For clarity, we will present below the main results for the value function, then we will shift to the action-value function when presenting our model.

III. Towards Our Model

Our work uses techniques developed for the problem of online on-policy evaluation, where an approximate action-value function is maintained and improved after each time step of following the policy. In particular we are interested in a linear Q-function approximator that uses Temporal Difference learning (TD) [12] since TD learning can be guaranteed to converge with any linear function approximator and suitable step size [13]. For the continuous case and non-linear function approximation, convergence is not guaranteed [14] although some models have been presented with good results [15]

In this work we focus on presenting a model that learns an approximation of a policy's action-value function from sample trajectories of experience following that policy. A method for solving this problem is a core component of our visual robot homing model. In particular, maintaining an online estimate of the Q-function can be combined with generalized policy improvement (GPI) to learn a controller [11].

For a particular value function V let the TD error at time t be defined as:

$$\delta_t(V_\theta) = r_{t+1} + \gamma V(s') - V(s) \quad (5)$$

$$\delta_t(V_\theta) = prediction_{t+1} - prediction_t$$

Then, $E_t[\delta_t(V^{\pi})] = 0$, that is, the mean TD error for the policy's true value function must be zero. We are interested in approximating V^{π} using a linear function approximator. In particular, suppose we have a function which gives a feature representation of the state space $\phi : S \rightarrow \Re^n$. We are interested in an approximated value function of the form $V_\theta = \phi(s)^T \theta$; $\theta \in \Re^n$ are the parameters of the value function.

Because the policy's true value function may not be in our space of linear functions, we want to find a set of parameters that approximates the true function. One possible approach is to use the observed TD error on sample trajectories of experience to guide the approximation.

The standard one-step TD method for value function approximation is TD(0). The basic idea of TD(0) is to adjust the predicted value of a state to reduce the TD error. Given some new experience tuple $(s_t, a_t, r_{t+1}, s_{t+1})$, the update with linear function approximation is:

$$\theta_t = \theta_{t+1} + \alpha_t u_t(\theta_t) \quad (6)$$

$$u_t(\theta_t) = \delta_t(V_\theta)\phi(s_t) \quad (7)$$

V_θ is the estimated value with respect to θ_t and α_t is the learning rate. The vector $u_t(\theta_t)$ is like a gradient estimate that specifies how to change the predicted value of s_t to reduce the observed TD error. We will call $u_t(\theta_t)$ the TD update at time t. After updating the parameter vector, the experience tuple is removed form memory.

IV. The Proposed Visual Homing Sarsa Model

In this section we describe the proposed model. In the simplest perspective, any reinforcement learning model, (or any MDP model in general), consists of elements and experience gained about those elements. The environment dynamics encoded in the tuple $(S, A, P^a_{ss'}, \Re^a_{ss'}, \gamma)$ describes the basic elements of the model, while the interaction between the robot and the environment constitutes the gained experience. This experience is normally encoded in the learning parameters using some learning method that mainly learns a value function. For control, reaching an optimal policy π^* can be done through policy improvement. We first begin by describing the main elements, then we describe the learning rules and algorithm, and conclude this section with the overall model structure.

A. Basic Elements of the System, the State Space:

Since we are considering *visual* homing, it is natural to choose the vision as the main medium to distinguish between different situations. Hence, we assume it is the image at each time step that represents the current state, and the state space ***S*** is the set of all the images that can be possibly taken for any location (with specific orientation) in the environment. This complex state space has two problems. First, each state is of high dimensionality, i.e. each state is represented by a large number of pixel components. Second, this 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 images state space and to gain the advantages of coding [16].

This feature representation of state space is assumed to reserve the distinctiveness of states, hence it can reduce the high-dimensionality problem but we are still faced by the intractability problem. Therefore, a generalization technique is needed in order to accommodate the intractability of state space. More precisely, generalization is needed in order to approximate the value for a state that has never been visited before, through previous visits to a similar states. A natural way to do so is to use a function approximation technique such as a neural network.

We would like to encode in those features implicitly how different the current image view is from those of the goal. This visual clue should guide the process of finding the goal location. The problem is that this approach does not give a direct distance indication. We will not assume that the goal location is always in the robot's field of view, but by comparing the current view with the goal view we combine the properties of distinctiveness, distance and orientation in one representation.

B. Defining the goal location:

Since the home location can be approached from different directions, the way it is represented should accommodate this fact. Therefore, a home (or a goal) location is defined by *m* snapshots called the *stored views*. The few snapshots (normally $m \geq 3$) of the home location are taken at the very start, each from a fixed distance but from a different angle. The distance should be compatible with the scale of the environment and the characteristics of the home location. This allows for the highest distinctiveness of the location without loosing info or involving unneeded information. These snapshots are the *only requirement* of the system to learn to reach its home location starting from any position in the environment (including those from which it cannot see the home from, i.e. the robot should be able to reach a hidden goal location).

C. The Features Vectors:

We take a histogram of each channel of the current view and compare it with those of the stored views through a radial basis function (RBF) layer. This gives us the feature space $\Phi : S \rightarrow \Re^n$ representation (8) which is used with the Sarsa(λ) algorithm, as we shall see later.

$$\phi_i(s_t(c,j)) = \exp(-\frac{\left\| h_i(s_t(c)) - h_i(v(c,j)) \right\|^2}{2\sigma_i^2}) \quad (8)$$

The index ***t*** is for the time step, ***j*** stands for the ***j***th stored view, and c is the index of the channel where we used the RGB representation of images. So $v(c,j)$ is the image of channel c of the ***j***th stored view, $h_i(v(c,j))$ is the bin *i* of the image $v(c,j)$, and $h_i(s_t(c))$ is bin *i* of the channel *c* of the current (*t*) view. Of course the number of bins has an effect on the performance of this measure and hence on the model, and will be studied in the experimental section.

D. The Action Space:

The set of actions is ***A*** = [Left_Forward, Right_Forward, Go Forward], where the two differential wheel speeds were set to a fixed values so that we have a countable set of actions.

E. Dissimilarity Measure and The Termination Condition:

We need a way to determine how close the current position is to the goal location, this is done through measuring how *dissimilar* is the *current view* to each *stored view* of the goal location. One can use any of the dissimilarity measures discussed extensively in the information retrieval field [10, 17]. In particular we are interested in the Jeffery Divergence Measure, given by (9).

$$JDM^{(H,K)} = \sum_i \left(h_i \log\frac{2h_i}{h_i + k_i} + k_i \log\frac{2k_i}{h_i + k_i} \right) \quad (9)$$

Where *H* and *K* are two images to be compared, h_i and k_i are the number of elements belong to bin *i* of the histograms of *H* and *K*, respectively. Fig. 1 (a) shows a simple view of robot's camera, part (b) shows the changes that took place in JDM measurements when turning away from this location.

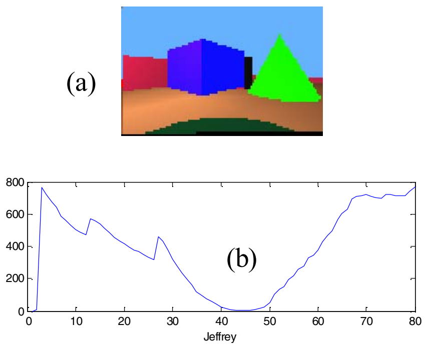

Fig. 1. Example of the JDM behaviour relative to the robot rotational motion

JDM has been successfully used with omni-directional camera to perform robot localization [10]. We used a normal camera, however, to be able to distinguish the robot's orientation which is crucial to our navigational task. This is to avoid the disadvantage of orientation-insensitivity of omni-directional camera which is desirable for localization but undesirable for navigation.

We will denote $JDM_t(c,j)$ as being the Jeffery Divergence Measure between the current view and the stored view *j* according to the channel *c*, and we denote $\overline{JDM}_t(j)$ to be the average dissimilarity between the current view and the stored view j on all of the channels:

$$\overline{JDM}_t(j) = \sum_c JDM_t(c,j) \Big/ |C| \quad (10)$$

We set our termination state to be the current view for which one of its $JDM_t(c,j)$ with the *m* stored views is less than a certain threshold ψ, i.e. the view that matches 'well' with one of the goal views.

$$\textit{If} \quad \min_{c,j}(JDM_t(c,j)) < \psi \qquad \Rightarrow \textit{Terminate Episode.}$$

The way to set this environment-scale-specific threshold is discussed in the experimental section.

F. The Reward Function:

The reward function $\Re^a_{ss'}$ consists of three parts:

-The main part is the *cost* which is set to -1 for each step taken by the robot without reaching the home location (reaching a termination state).

The reward signal can be augmented by another two signal to insure higher performance although the model works regardless of their involvement. Those are:

-Approaching the goal reward: is the maximum *reduction in dissimilarity* between the current step and the previous step. If this difference is decreasing it means that the robot is actually moving in the right direction towards the home location. While if it is increasing it means the opposite. We call this signal the *differential dissimilarity signal* and it is defined as:

$$\Delta\overline{JDM_t} = \max_j(\overline{JDM_{t-1}}(j) - \overline{JDM_t}(j)) \quad (11)$$

- The Position signal is the inverse of the current dissimilarity. $\frac{1}{\overline{JDM_t}}$

Thus, as the current location differs less, from the home location, this reward will increase.

$$r = \cos t + \Delta\overline{JDM_t} + \frac{1}{\overline{JDM_t}} \quad (12)$$

Of course the previous two reward component will only be considered if the dissimilarities of both steps falls under a certain threshold ψ' to ensure that the robot is approaching the home location. This threshold is environment- scale-specific, and is introduced merely to enhance the performance.

The overall structure of the model is shown in Fig. 2.

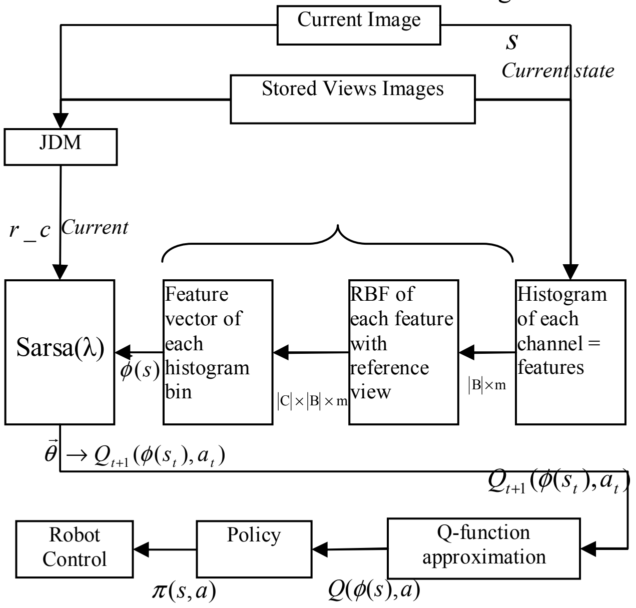

Fig. 2. The various component of the proposed model.

G. *The Eligibility Trace:*

An eligibility trace constitutes a mechanism for temporal credit assignment. It marks the memory parameters associated with the action as eligible for undergoing learning changes [11]. Depending on our application, the eligibility trace for action *a* is the discounted sum of the feature vectors for the images that the robot has seen so far, after applying this action. The eligibility trace for other actions which has not been taken while in the current state is simply its previous trace but discounted, i.e. those actions are now less responsible for the credit:

$$\vec{\mathbf{e}}_t(a) \leftarrow \begin{cases} \gamma\lambda\vec{\mathbf{e}}_{t-1}(a) + \vec{\boldsymbol{\varphi}}(s_t) & \text{if } a = a_t \\ \gamma\lambda\vec{\mathbf{e}}_{t-1}(a) & \text{otherwise} \end{cases} \quad (13)$$

where λ is the discount rate for the eligibility traces $\vec{\mathbf{e}}$

H. *The Learning Method:*

The remaining is the learning algorithm. Our algorithm is an on-policy bootstrapping Sarsa(λ) [11] with linear approximation of the Q action-value function. Sarsa(λ) is an algorithm that uses TD(λ) for control. It learns on-line through interaction with a simulation software that feeds 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 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 gives 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 GPI. The overall algorithm is that of the Sarsa(λ) control algorithm [11] and is summarized in Fig. 3.

```
*Initialization*
$\vec{\boldsymbol{\theta}}_0(a_i) = \vec{1} \qquad i = 1:|A|$
$a_0 \leftarrow 2$
Repeat for each episode
  $\vec{\mathbf{e}}_0(a_i) = \vec{\mathbf{0}} \qquad i = 1:|A|$
  $s_0 \leftarrow$ 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 $\pi(\vec{\boldsymbol{\varphi}}(s_{t+1}), a)$.
    $\delta_t \leftarrow \left[r_{t+1} + \gamma\vec{\boldsymbol{\varphi}}(s_{t+1})^T \cdot \vec{\boldsymbol{\theta}}(a_{t+1}) - \vec{\boldsymbol{\varphi}}(s_t)^T \cdot \vec{\boldsymbol{\theta}}(a_t)\right]$
    $\vec{\mathbf{e}}_t(a) \leftarrow \begin{cases} \gamma\lambda\vec{\mathbf{e}}_{t-1}(a) + \vec{\boldsymbol{\varphi}}(s_t) & \textit{if } a = a_t \\ \gamma\lambda\vec{\mathbf{e}}_{t-1}(a) & \textit{otherwise} \end{cases}$
    $\vec{\boldsymbol{\theta}}(a_t) \leftarrow \vec{\boldsymbol{\theta}}(a_t) + \alpha_{ep} \cdot \vec{\mathbf{e}}_t(a_t) \cdot \delta_t$
    $\vec{\boldsymbol{\varphi}}(s_t) \leftarrow \vec{\boldsymbol{\varphi}}(s_{t+1})$
    $a_t \leftarrow a_{t+1}$
  until $\min_j(\overline{JDM_t}(j)) < \psi_1$
until *episode == final_episode*
```

Fig. 3. Linear on-policy gradient-descent Sarsa(λ) control, with RBF features algorithm for linear action-value function approximation and Policy Improvement. The approximate Q is implicitly a function of $\vec{\theta}$

The learning rate was the same used by Boyan [18]:

$$\alpha_{ep} = \alpha_0 \cdot \frac{n_0(\textit{final\_episode}) + 1}{n_0(\textit{final\_episode}) + \textit{episode}} \quad (14)$$

I. *The policy used to Generate Actions:*

A combination of ε-greedy policy and Gibbs soft-max [11] policy is used to pick an action and to strike the balance between exploration and exploitation. Using ε-greedy probability allows exploration to be increased as needed by initially setting ε to high value then decreasing it through episodes. Gibbs soft-max probability,

$$Gibbs\ (a_{(i)}, \vec{\varphi}(s_t)) = \frac{\exp\left[\vec{\varphi}(s_t)^T \cdot \vec{\theta}(a_{(i)})\right]}{\sum_{j=1}^{|A|} \exp\left[\vec{\varphi}(s_t)^T \cdot \vec{\theta}(a_{(j)})\right]}, \quad (15)$$

helped in increasing the chances of picking the action with the highest value when the differences between the values of it and the remaining actions is large, i.e. it helped in increasing the chances of picking the action with the highest Q-value when the robot is sure that it is the right one.

$$\Pr(a, \vec{\varphi}(s_t)) = \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|A|} & if\ \ a = \arg\max_i \left[\vec{\varphi}(s_t)^T \cdot \vec{\theta}(a_{(i)})\right] \\ \dfrac{\varepsilon}{|A|} & otherwise \end{cases} (16)$$

$$\pi_{\varepsilon+Gibbs}(a_i, \vec{\varphi}(s_t)) = Gibbs(a_i, \vec{\varphi}(s_t)) + \Pr(a_i, \vec{\varphi}(s_t)) \quad (17)$$

J. The Neural Network Layers:

From a neural network point of view, when considering the RBF layer together with the competitive layer, one can realize that this architecture is similar to a Probabilistic Neural Network (PNN) that calculates the probability of picking up a certain action conditional to the given goal. We will call the neural network used in our model the RBF-Q-D Network (and algorithm) because we used the RBF layer for feature extraction and then a linear layer with Sarsa(λ) algorithm and the dissimilarity measure. Fig. 3 shows a simplification of our model with its layers.

K. The Linear Networks and Features Dimensions :

The parameters have the same dimension as the feature space which is $n = |C| \times |B| \times m$; where $|C| = 3$ is the number of channels, $|B|$ is the number of bins per image and m is the number of stored views for the goal location. Since we use an RGB images with values in the range of [0 , 255] for each pixel, the dimension of the feature space is given by:

$$n \approx |\mathrm{C}| \times \frac{256}{b} \times m \quad (18)$$

where b is the bin's size. Different bin sizes give different dimensions, which in turn give a different number of approximation parameters θ. The equality is not complete due to the fact that the precise number of bins is going to be $|\mathrm{B}| = round(\frac{256}{b}) + 1$.

Note that σ_i of the features has been chosen through continuous update of the sum of the features vector collected in all the time steps so far.

$$\sigma_i = \left(\frac{1}{|T| - 1} \cdot \sum_T \left\| h_i(s_{t+1}(c, j)) - h_i(v(c, j)) \right\|^2 \right)^{1/2} \quad (19)$$

This allowed for maintaining better incremental estimation of each feature variance and hence better performance. After enough exploration of the environment this value is almost stable and changes to it are minimized. It has been observed that the variance of this internal parameter has dropped after *episode 10* to a negligible value, which means results are reliable for *episode>10* and that the neural networks are learning the value function for almost the same states that are going to be encountered in the future.

L. Important enhancements and Limitations:

One problem of '*unnecessary wandering*' remains. Mainly it is caused by consequent conflicting positive and negative rewards given by the environment due to approaching the goal and wandering around it without reaching it. Simply a sever punishment was applied for the particular case when the robot goes from a positive rewarding to a negative punishments in two successive steps.

V. EXPERIMENTAL RESULTS

The model was applied using a simulated *Khepera* [19] robot in *Webots*™ [20] simulation software. The Khepera is a miniature real robot, 70 mm diameter and 30 mm height, and is provided with 8 infra-red sensors for reactive behaviour, as well as a colour camera extension.

A 1.15x1 m^2 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.

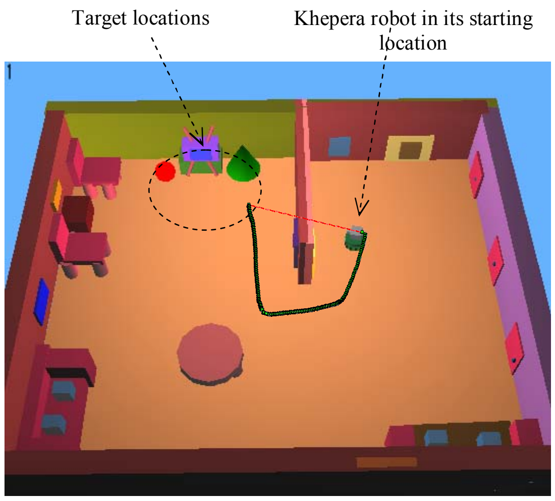

Fig. 4. Snapshots of the realistic simulated environment.

Fig. 4 shows the environment used. A cone, ball and TV are included to add more texture to the goal location, i.e. to enrich it and make it different from the other environment locations. We reemphasize that no object recognition technique was used, only the JDM. The controller written as a combination of C++ code and Matlab Engine code.

The robot starts by taking (*m*= *3*) snapshots for the goal location. It then goes through a specific number (500) of episodes. The robot starts with a random policy, and finishes an episode when it reaches the desired location.

A. The Practical Settings of the Model Parameters:

For our application we have chosen the feature space parameters to be *b=3, m=3* hence

$n = 3\times(round(256/3)+1)\times 3 = 774$. λ was set to the value of 0.8 depending on the studies [11, 21] that referred to the range of [0.7 0.8] as the peak of the performance of the TD(λ)-learning. The discount constant was set to $\gamma = 1$, i.e. there is no discount through time. ψ, ψ' are purely empirical and were set to 1.7 and 2 respectively.

B. Setting the Exploitation vs. Exploration:

Since action space is finite, and to avoid fluctuation and overshooting 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 are sufficiently visited. Setting exploration high also helps in decreasing the number of episodes needed before reaching an acceptable performance. This explains the exponential appearance of the different learning curves discussed below.

The model performance has been studied for a small number of stored views (*m=3)* to show the robustness of the model. One can enhance accuracy by increasing the dimension space but would have to trade-off speed of convergence and execution. The most natural way to increase the state space dimension is by increasing the number of histogram's bins considered. However, to concentrate on the pure effect of changing *m* and eliminate the increase in state dimension due to the increase in *m* (18)**,** one can set *m=b* then change both *m* and *b* together. This could fix the dimension of the feature space and consequently the size of the approximator, and show the actual effect of changing the number of views *m*.

C. Studying the Model Performance:

Fig. 5, shows the effect of learning averaged over 8 trials, each with 500 episodes. All of the trials successfully converged. Divergence occurred only when setting the learning rate to a high value, or when exploration was quickly decreased. The reason that we needed a low learning rate is that we use a Gibbs probability distribution for the exploration/exploitation balance. This exponentially formed probability can go quickly to infinity if care is not taken when assigning its exponents. The fact that we have a relatively large state space dimension was the major factor in this situation.

Part (a) shows the most important aspect of any reinforcement learning model; the return values of each episode, converging optimally. After all, the main purpose of the RL-based model is to optimally increase the sum of the received rewards. The return values (mostly negative) have increased naturally through episodes due to the improvement taking place from episode to episode. This is done via improving the adopted policy implicitly; by moving to better estimates and decreasing exploration from episode to the other. The accuracy of the action-value function estimates is gradually/iteratively increasing using the learning parameter $\vec{\theta}$.

Fig. 5 (b) shows the decrease that took place in the number of steps needed to achieve the task. This normal decrease is in accordance with part (a) and because of the cost part of the reward function. In fact, we decreased the difference between ψ and ψ', so that the other two parts of the reward formula have minimal effect on the model convergence. Fig. 5 (c) depicts the changes in the learning parameters themselves, *i* stands for the component index of the learning vector.

Most important is that the three parts have an exponential-like shape showing the high speed of convergence this model has reached. This is highly desirable in reinforcement learning model due to its dominant convergence slowness problem [11]. In fact, one major contribution of this work is the high performance reached with little experience using a complex visual input.

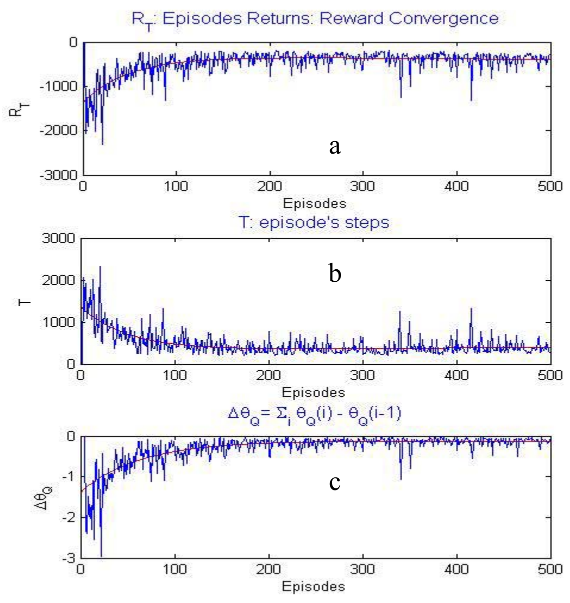

Fig. 5. Learning Curves averaged over 8 trials.

Fig. 6, depicts performance and internal parameters illustrations. Fig. 6, (a) shows the learning rate decrease through the episodes which was used throughout the trials. Part (b) shows the decrease enforced on the exploration rate ε while part (c) shows the overall percentage of explorative action and exploitative actions. Routes taken by the robot in three episodes (early, middle, and final) for one of the trials are shown in parts (d)-(f).

VI. Discussion And Conclusion

We have tackled the policy improvement for Sarsa(λ) systems combined with JDM and RBF. This is novel to models introduced in the literature due to the way we applied reinforcement learning using neuro-dynamic programming methods like Sarsa(λ). Below we state some of the advantages of this model:

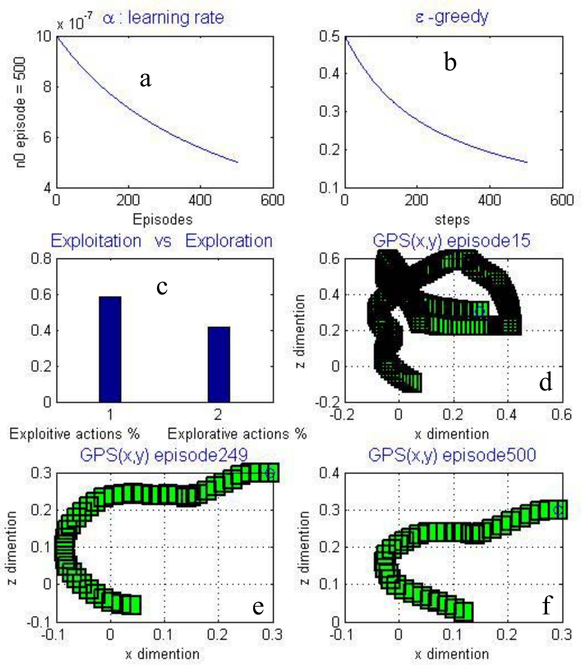

Fig. 6. Learning performance and a sample route for a sample trial.

1) Simplicity of learning: the robot can learn to perform its visual navigation task in a very simple way without a long process of map building.
2) Limited storage of information is required in the form of *m* stored views.
3) No pre or manual processing is required. No *a priori* knowledge about the environment is needed i.e. no landmarks are needed in those views.
4) An important advantage of our model over MDP explicit model-based approaches is that abduction of robot is solved directly i.e. the robot can find its way and recover after it has been displaced from its current position and put in a totally different position.

To raise the differentiability of the views, however, they should be rich with colours etc. (i.e. good amount of information).

Through the learning robustness and generality of RL robots, coupled with *visual* states and rewards, the system achieved a high level of robustness, generality, and applicability. This combination tentatively proved to work very well for our navigation problem.

Future work includes carrying out more extensive experiments over our model by trying different configurations using (18), both in terms of more views to be considered as well as different bins sizes and different environments. Future work can also include using off-policy instead of the on-policy method to accommodate for two behaviours layers used by the agent.

References

[1] U. Nehmzow, *Mobile robotics: A Practical Introduction*: Springer-Verlag, 2000.

[2] A. M. Anderson, "A model for landmark learning in the honey-bee," *Journal of Comparative Physiology A* vol. 114, pp. 335-355, 1977.

[3] B. A. Cartwright and T. S. Collett, "Landmark maps for honeybees," *Biological Cybernetics*, vol. 57, pp. 85-93, 1987.

[4] A. Vardy and F. Oppacher, "A scale invariant local image descriptor for visual homing," in *Biomimetic neural learning for intelligent robots.* , G. Palm and S. Wermter, Eds.: Springer, 2005

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

[6] D. Sheynikhovich, R. Chavarriaga, T. Strosslin, and W. Gerstner, "Spatial Representation and Navigation in a Bio-inspired Robot," in *Biomimetic Neural Learning for Intelligent Robots*, S. Wermter, M. Elshaw, and G. Palm, Eds.: Springer, 2005, pp. 245-265.

[7] C. Weber, D. Muse, M. Elshaw, and S. Wermter, "A camera-direction dependent visual-motor coordinate transformation for a visually guided neural robot," *Knowledge-Based Systems, Science Direct, Elsevier* vol. 19, pp. 348–355, 2006.

[8] S. Thrun, Y. Liu, D. Koller, A. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *International Journal of Robotics Research*, vol. 23, pp. 693–716, 2004.

[9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Massachusetts; London, England: The MIT Press, 2005.

[10] I. Ulrich and I. Nourbakhsh, "Appearance-Based Place Recognition for Topological Localization," presented at IEEE International Conference on Robotics and Automation, San Francisco, CA, 2000.

[11] R. S. Sutton and A. Barto, *Reinforcement Learning, an introduction*. Cambridge, Massachusetts: MIT Press, 1998.

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

[13] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control* vol. 42, pp. 674–690, 1997.

[14] J. A. Boyan, "Technical update: Least-squares temporal difference learning. ," *Machine Learning* vol. 49., pp. 233–246, 2002.

[15] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Q-Learning in Continuous State and Action Spaces," presented at Australian Joint Conference on Artificial Intelligence, Australia, 1999.

[16] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for robocup soccer keepaway," *International Society for Adaptive Behavior* vol. 13, pp. 165–188, 2005.

[17] Y. Rubner and et al., "The Earth Mover's Distance as a Metric for Image Retrieval," *International Journal of Computer Vision*, vol. 40, pp. 99-121, 2000.

[18] J. A. Boyan, "Least-squares temporal difference learning. ," presented at In Proceedings of the Sixteenth International Conference on Machine Learning, San Francisco, CA, 1999.

[19] D. Floreano and F. Mondada, "Hardware solutions for evolutionary robotics," presented at First European Workshop on Evolutionary Robotics, Berlin, 1998.

[20] O. Michel, " Webots: Professional Mobile Robot Simulation," *International Journal of Advanced Robotic Systems*, vol. 1, pp. 39-42, 2004.

[21] 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.