Robot Docking Based on Omnidirectional Vision and Reinforcement Learning

David Muse, Cornelius Weber and Stefan Wermter
Hybrid Intelligent Systems, School of Computing and Technology
University of Sunderland, UK. Web: www.his.sunderland.ac.uk

Abstract

We present a system for visual robotic docking using an omnidirectional camera coupled with the actor critic reinforcement learning algorithm. The system enables a PeopleBot robot to locate and approach a table so that it can pick an object from it using the pan-tilt camera mounted on the robot. We use a staged approach to solve this problem as there are distinct sub tasks and different sensors used. Starting with random wandering of the robot until the table is located via a landmark, and then a network trained via reinforcement allows the robot to turn to and approach the table. Once at the table the robot is to pick the object from it. We argue that our approach has a lot of potential allowing the learning of robot control for navigation removing the need for internal maps of the environment. This is achieved by allowing the robot to learn couplings between motor actions and the position of a landmark.

1 Introduction

Navigation is one of the most complex tasks currently under development in mobile robotics. There are several different components to navigation and many different sensors that can be used to complete the task, from range finding sensors to graphical information from a camera. The main function of robot navigation is to enable a robot to move around its environment, whether that is following a calculated or predefined path to reach a specific location or just random wandering around the environment. Some of the components involved in robotic navigation are *(i)* localisation, *(ii)* path planning and *(iii)* obstacle avoidance. For an overview of localisation and map-based navigation see [1 & 2]. When discussing robot navigation, simultaneous localisation and map building should be included (see [3, 4 & 5] for some examples).

There has been a lot of research and systems developed for robot navigation using range finding sensors (sonar, laser range finders etc) [6, 7 & 8] but there has been less research into visual robotic navigation. There are recent developments in the field of visual navigation mainly concentrating on omnidirectional vision (see [9, 10 & 11] for examples).

Many of the navigation systems implemented for robot navigation still use hard coding which causes a problem with the lack of adaptability of the system. However, some systems have included learning (see [12 & 13] for examples). A common training method used for the learning systems are various forms of reinforcement learning, [14] provides a good overview. These learning algorithms overcome the problem of supervised learning algorithms as input output pairs are

not required for each stage of training. The only thing that is required is the assignment of the reward which can be a problem for complex systems as discussed in [15]. However, for systems where there is just one goal this does not pose a problem as the reward will be administered only when the agent reaches the goal.

The focus on this paper is to extend the system developed in [16] where reinforcement learning is used to allow a PeopleBot to dock to and pick an object (an orange) from a table. In this system neural vision is used to locate the object in the image, then using trained motor actions (via the actor critic learning algorithm [17]) the aim is to get the object to the bottom centre of the image resulting in the object being between the grippers of the robot.

There are some limitations to the system which need to be overcome to improve its usefulness. For example, the docking can only work if the object is in sight from the beginning which results in the system being confined to a very small area. Also the system fails if the object is lost from the image. Finally, the angle of the robot with respect to the table is inferred from the odometry, which makes it necessary to start at a given angle. None of these are desirable and it is the aim of this work to address some of the limitations and extend the range that the robot can dock from.

The system proposed in this paper will make use of an omnidirectional camera to locate and approach a table in an office environment. The use of an omnidirectional camera allows the robot to continuously search the surrounding environment for the table rather than just ahead of the robot. Here the extended system will use the omnidirectional camera to locate the table via a landmark placed beneath it. Once located the robot is to turn and approach the table using a network trained by reinforcement.

The remainder of the paper is structured as follows; Section 2 discusses the task, the overall control of the system and what triggers the shifts between the different phases. The first phase uses an omnidirectional camera to detect any obstacles and take the necessary action to avoid them and is discussed in Section 2.1. The second phase uses the omnidirectional camera to locate the position of the landmark in relation to the robot, which it then passes to a neural network to produce the required motor action on the robot and is discussed in Section 2.2. The final phase uses a neural system with the pan tilt camera mounted on the robot to allow the robot to dock with the object on the table and pick it up; this is discussed in Section 2.3. Section 3 covers the algorithm used for the table approaching phase of the extended scenario. The experimentation of the extended scenario is then described in section 4. Finally, Sections 5 and 6 cover the discussion and summary respectively.

2 The Scenario

The overall scenario is illustrated in Figure 1. It starts with the robot being placed in the environment at a random position away from the table. The robot is then to wander around the environment until it locates the table (Phase I). This phase uses conventional image processing to detect and avoid any obstacles. Once the table is located via a landmark placed beneath it the robot is to turn and approach the table

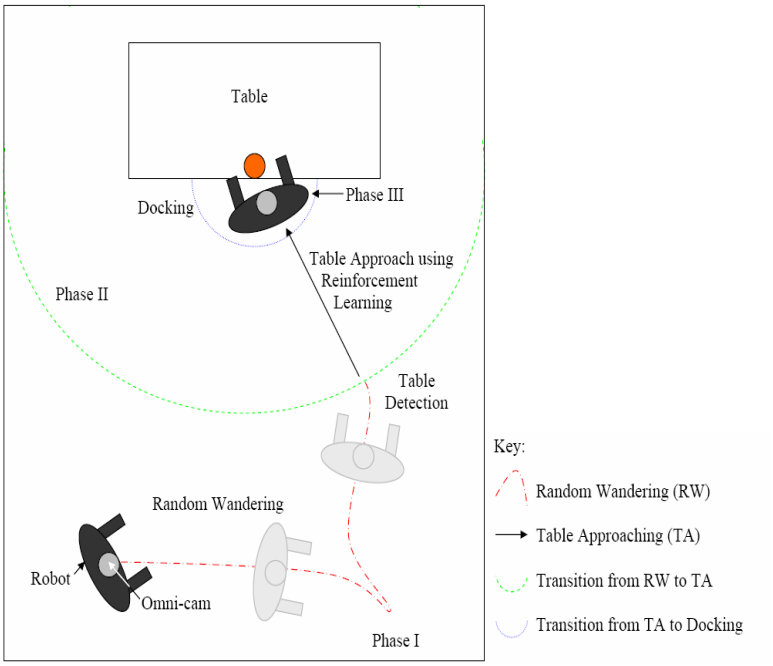

Figure 1 - Scenario

(Phase II). Then once at the table the robot is to pick the object from the table (Phase III), this system is discussed in [16]. Both Phase II & III use neural networks trained with the Actor Critic learning algorithm.

The first two phases of the system use an omnidirectional camera illustrated in Figure 2 and the final phase uses the pan tilt camera mounted on the robot.

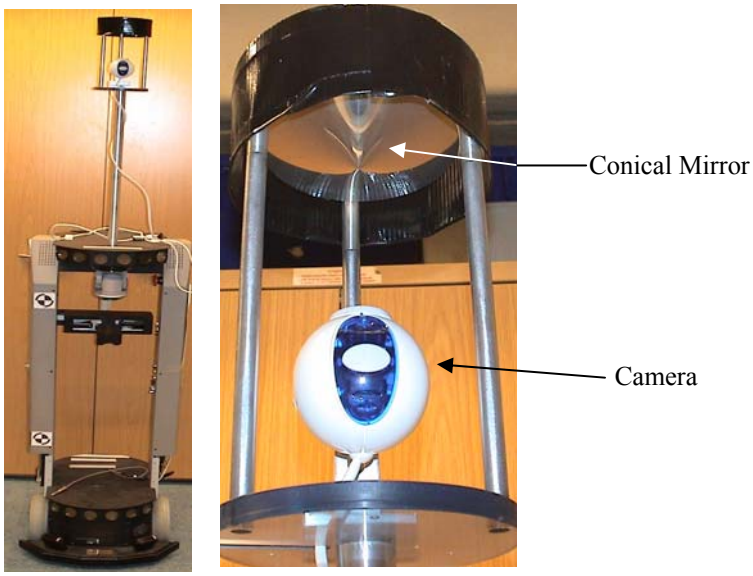

Figure 2 – (Left) PeopleBot Robot with mounted omnidirectional camera, (Right) Close up of the Omnidirectional Camera

To enable the integration of the three phases an overall control function was needed to execute the relevant phases of the system depending on the environmental conditions. Figure 3 shows the control algorithm.

```
While the robot is not at the table
        Take a picture (omnidirectional)
        Check if the landmark is in sight
        If the landmark is not in sight
                Wander
        Else the landmark is in sight
                Pass control to the actor critic and get exit status
                If exited because landmark is lost
                        Go back to Wandering
                Else exited because robot is at the table
                        Pass control to the object docking
                End if
        End if
End While
```

Figure 3 - System Algorithm

When the robot is not at the table, or the landmark is not in sight, the robot checks for the landmark at each iteration through the system. The landmark that the robot looks for is produced by a board of red LED's which is located directly beneath the table as illustrated in Figure 4.

While the robot has not located the landmark the random wandering system is executed. If the landmark has been located then control is passed to the table approaching behaviour which runs to completion. There are two possible outcomes for the table approaching which are; *(i)* Lost sight of the landmark and *(ii)* Reached the table. If the landmark has been lost then the robot starts to search for it again, otherwise it has reached the table and control is passed to the object docking which completes the task.

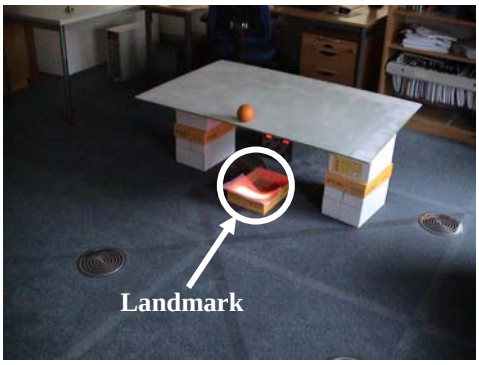

Figure 4 - Setup of the Environment: The landmark is used to identify the position of the table

2.1 Phase I – The Random Wandering

This behaviour allows the robot to move around the environment while avoiding obstacles. The system uses an omnidirectional camera (Figure 2 right) to get a view of the environment surrounding the robot. From this image the robot is able to detect obstacles and produce the required motor action to avoid them. To perform this detection the behaviour uses classical image processing to remove the background from the image and leave only perceived obstacles, as seen in Figure 5. Here the original image taken by the omnidirectional camera is in the left of the figure, with different stages of the image processing shown in the centre and right.

Figure 5 – Obstacle Detection

The centre image is the intermediate stage where just the background of the image has been removed; this is achieved by colour segmentation of the most common colour from the image. To find the most common colour in the image a histogram is produced for the RGB values of each pixel. Then the value with the largest density is found and any colour within the range of +/- 25 of the most common colour is removed. This removes the carpet from the image (assuming that the carpet is present in the majority of the image) which leaves the obstacles and some noise. Also at this stage the range of the obstacle detection is set removing any noise from the periphery of the image. Then the noise is removed by image erosion followed by dilation. The erosion strips pixels away from the edges of all objects left in the image. This removes the noise but it also reduces the size of any obstacles present. To combat this once the erosion has been performed, dilation is performed to restore the obstacles to their original size, the shape of the obstacles are slightly distorted by this process. However, the obstacles left in the final image are still suitable to produce the required motor action to avoid them. The last stage of the image processing is to use edge detection to leave only the outlines of the obstacles (Figure 5 right).

The robot always tries to move straight ahead unless an obstacle is detected in the robot's path. When this happens the robot turns the minimum safe amount allowed to avoid the obstacles. In the example provided in Figure 5, the robot cannot move straight ahead so the robot would turn to the left until it can avoid the obstacle on the right of the image. As the image is a mirrored image of the environment the objects which appear on one side of the image are physically to the other side of the robot. Once the robot has turned the required amount it would start to move straight and the obstacle detection would then be performed again.

2.2 Phase II – The Table Approaching Behaviour

This phase of the system allows the robot to approach the table (landmark) once detected. This has two exit statuses which are *(i)* the robot lost sight of the

landmark or *(ii)* the robot has reached the table. If the robot looses sight of the table it goes back to the wandering phase until it locates the landmark again. This can happen if the landmark moves behind one of the supporting pillars of the conical mirror. If the robot reaches the table, control will be passed to the final stage of the system which is to dock to and pick up the object.

To allow the robot to move to the table a network was trained using the Actor Critic reinforcement learning rule [17]. The state space was the image with the goal set to where the landmark is perceived to be in front of the robot. The motor action that the network performs is to rotate the robot to the left or to the right depending on where the landmark is perceived in relation to the robot. The input to the network is the x y coordinates of the closest point of the perceived landmark. Once the landmark appears to be ahead of the robot, the robot then moves forward, checking that the landmark is still ahead of it. Once the landmark is ahead of the robot and less than the threshold distance of 1 meter the robot then moves directly forward until the table sensors located on the robot's base are broken. When this happens the robot is at the table and control is given to Phase III.

The robot only looks for the landmark in the range that the robot can detect directly ahead (as the webcam produces a rectangular image, more can be seen to the sides of the robot. The range is set to the maximum distance the image can detect ahead of the robot; this is roughly 2m). If the landmark is detected outside this range when the robot turned it would lose sight of the landmark, therefore anything outside this region is ignored. If the landmark appears in the right side of the detectable range then the robot should rotate to the left as the image is mirrored, if it appears in the left the robot should rotate to the right and if it is straight ahead of the robot then it should move forward.

Figure 6 - Landmark Detection

To detect the landmark classical image processing is once again employed to detect the landmark as shown in Figure 6. The original image is in the left of Figure 6 with the landmark highlighted and the detected landmark is highlighted in the right of Figure 6. The first stage to the image processing is to perform colour segmentation where it segments any colour that is the designated colour of the landmark. Once this process is complete edge detection is used to leave just the edges of the remaining objects. Then it is assumed that the largest object left in the image is the landmark. The last stage of the image processing is to locate the closest point of the landmark to the robot. This point is then fed into the network to produce the required action by the robot.

2.3 Phase III – Docking

This phase allows the robot to dock to and pick an orange from the table. The functionality of the system is described in [16]. However, there is a problem with this system for the integration into the extended scenario; the odometry of the robot is set to 0 and the robot must start parallel to the table to allow the robot to dock to the orange. With the table approaching system it cannot be guaranteed that the robot will be parallel to the table and hence the robot will not know the relationship between the odometry and the angle of the table.

Before this system is integrated it is required that the angle of the table to the robot is calculated. To solve this it is planned to use image processing to detect and calculate the angel of the table in relation to the robot. Once the robot reaches the table a picture will be taken using the conventional pan tilt camera mounted on the robot. The edge of the table will then be detected using colour thresholding and edge detection.

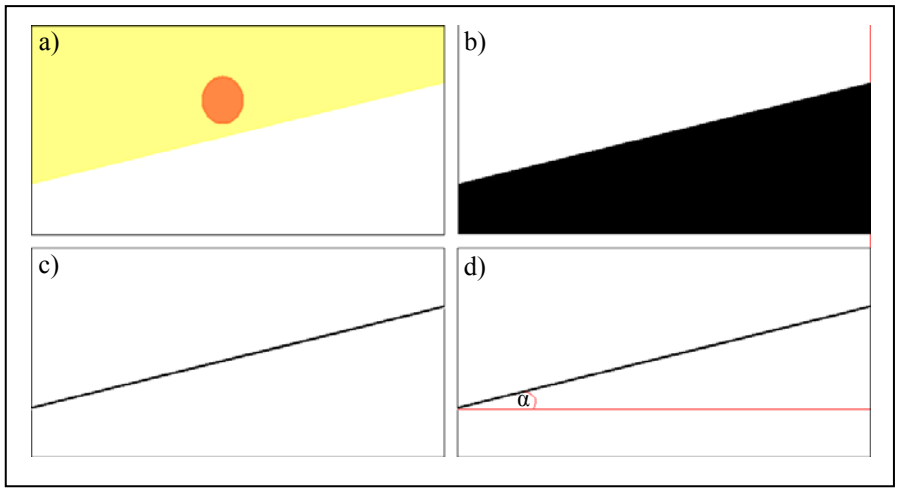

Figure 7 - Edge Detection of the Table

The thresholding will be performed in the same way as in Phase I with the most common colour being removed. It is assumed that the most common colour will either be *(i)* the colour of the table itself or *(ii)* the colour of the carpet beneath the table. In both cases the edge between the removed colour and the remaining colour will be the edge of the table. Using edge detection the coordinates of the two end points of this line can be found and from this the angle of the table calculated and used with the odometry to get the robot to dock to the orange.

Figure 7 demonstrates this image processing using the artificial image (a), here the white is thought to be the most common colour so will be removed and the remaining components of the image are changed to white (b). The next stage is to perform the edge detection (c). With this done the angle can be calculated (d) and used to alter the odometry of the robot. This is to remove the constraint that the robot must arrive parallel to the table.

3 Actor Critic Algorithm

The developed network is an extension of the actor critic model used in [17]. Here the system has been adapted to work with continuous real-world environments. We have used this algorithm in two phases of the scenario: first, the approach to the table (Phase II), and then to perform the docking at the object. In Phase II, the input to the network is the position in the omnidirectional image where the landmark appears as opposed to the location of the agent in the environment. In Phase III, the input is the perceived location of the object of interest from the standard robot camera.

For the architecture of the network developed for Phase II, it was decided that there would be two input neurons; one for the x and y coordinates respectively, 50 hidden units to cover the state space of the image and two output neurons one for each of the actions to be performed and one neuron for the critic. The architecture is illustrated in Figure 8. The hidden area covers only the detectable region of the image with each neuron covering roughly 40mm^2 of actual space. This results from the fact that the detectable range of the environment is roughly a radius of 2m from the robot. All units are fully connected to the hidden layer. Initially the critics' weights are set to 0 and are updated by Equation 4. The Actor weights (Motor Action units) are initialised randomly in the range of 0 – 1 and are updated via Equation 7. Finally, the weights connecting the input units to the network (High level vision) are set to 1 and these weights are not updated.

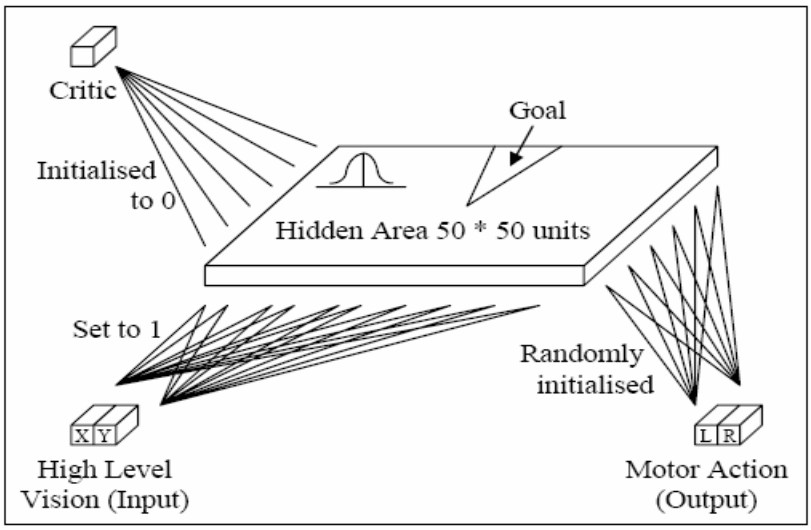

Figure 8 - Architecture of the Network. The nodes are fully connected, the input for the x, y coordinates are normalised into the range 0-50 and the output of the network generates the motor action to rotate the robot

Equation 1 describes the firing rate of the "place cells" (here the term place cell is used loosely as they encode a perceived position of a landmark in the image) to be calculated. The firing rate is defined as:

$$f_i(p) = \exp\left(-\frac{\|p - s_i\|^2}{2\sigma^2}\right) \quad (1)$$

where p is the perceived position of the landmark, s_i is the location in the image where neuron i has maximal firing rate and σ is the radius of the Gaussian of the firing rates covering the image space of each neuron. This was set to 0.75 during the experiments. The firing rate C of the Critic is calculated using Equation 2 and has only one output neuron as seen in Figure 8. The firing rate of the critic is thus a weighted sum of all of the firing rates of the place cells.

$$C(p) = \sum_i w_i f_i(p) \tag{2}$$

To enable training of the weights of the critic some method is needed to calculate the error generated by the possible moves to be made by the robot. This is made possible by Equation 3 and the derivation of this equation can be found in [17].

$$\delta_t = R_t + \gamma C(p_{t+1}) - C(p_t) \tag{3}$$

However as R_t only equals 1 when the robot is at the goal location and $C(p_{t+1})$ is 0 when this occurs and vice versa they are never included in the calculation at the same time. γ is the constant discounting factor and was set to 0.7 for the experiments. With the predicted error, the weights of the critic are updated proportionally to the product of the firing rate of the active place cell and the error (Equation 4).

$$\Delta w_i \propto \delta_t f_i(p_t) \tag{4}$$

This concludes the equations that were used for the place cells and the critic, finally there are the equations used for the actor. There were two output neurons used in this experiment, one to make the robot rotate to the left and the other to make the robot rotate to the right. The activation of these neurons is achieved by taking the weighted sum of the activations of the surrounding place cell to the current location as illustrated in Equation 5.

$$a_j(p) = \sum_i z_{ji} f_i(p) \tag{5}$$

A probability is used to judge the direction that the robot should move in, this is illustrated in Equation 6. Here the probability that the robot will move in one direction is equal to the firing rate of that actor neuron divided by the sum of the firing rate of all the actor neurons. To enable random exploration when the system is training, a random number is generated between 0 and 1. Then the probability of each neuron is incrementally summed; when the result crosses the generated value that action is executed. As the system is trained the likelihood that the action chosen is not the trained action decreases. This is because as the network is trained the probability that the trained action will occur will approach 1.

$$P_j = \frac{\exp(2a_j)}{\sum_k \exp(2a_k)} \tag{6}$$

Ultimately, the actor weights are trained using Equation 7 in a modified form of Hebbian learning where the weight is updated if the action is chosen and not updated if the action is not performed. This is achieved by setting $g_j(t)$ to 1 if the action is chosen or to 0 if the action is not performed. With this form of training both the actor and the critics weights can be bootstrapped and trained together.

$$\Delta z_{ji} \propto \delta_t f_i(p_t) g_j(t) \tag{7}$$

4 Experimentation and Results

To train and test the network separate training and test data sets were produced. The training set contained 1000 randomly generated samples and the test set contained 500 randomly generated samples. These samples were stored in separate vectors and contained the following information *(i)* the normalised x coordinate of the landmark, *(ii)* the normalised y coordinate of the landmark, *(iii)* the angle of the landmark in relation to the robot and *(iv)* the distance of the landmark from the robot. During training each sample was fed into the network and it ran until the goal was achieved. Therefore, after each epoch there would be 1000 successful samples and the testing data was fed into the network without any training taking place.

The trained weights of the critic are shown in Figure 9 (d), which took 50 epochs to get the training to the level shown. It would have been impractical to train the network on the robot due to the time it would require, so a simple simulator was employed which used the training set to perform the action recommended by the network (this used the same data that would be generated from the image processing). This was achieved by calculating the next perceived position of the landmark. This greatly reduced the time needed to train the network, for the 50 epochs it took roughly 5 hours to train (including the testing after each epoch) on a Linux computer with a 2GHz processor and 1 Gigabyte of ram. Figure 9 also shows the untrained weights (a), the weights after the presentation of 1 training sample (b) and the weights after the presentation of 500 training samples (c). Here it can be seen that the weights spread from the goal location around the network during the training. There is a 'V' section of the weights that remain untrained, this relates to the goal location (see Figure 8) so no training is needed in this section of the network as the required state is reached.

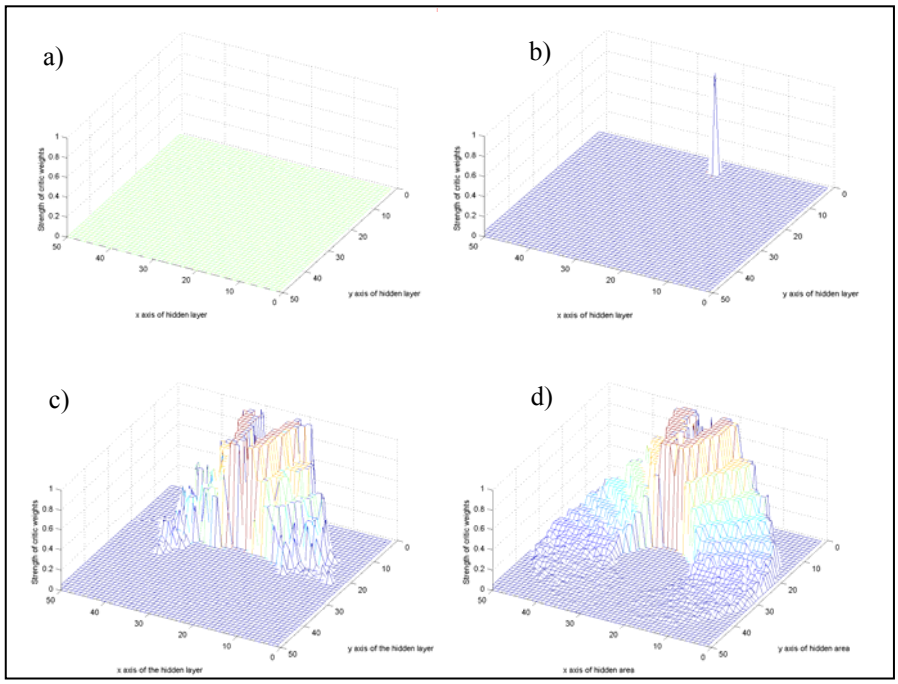

Figure 9 - Strength of Critic Weights During Training. (a) untrained weights, (b) weights after presentation of 1 sample, (c) weights after presentation of 500 samples and (d) weights after 50 epochs of training

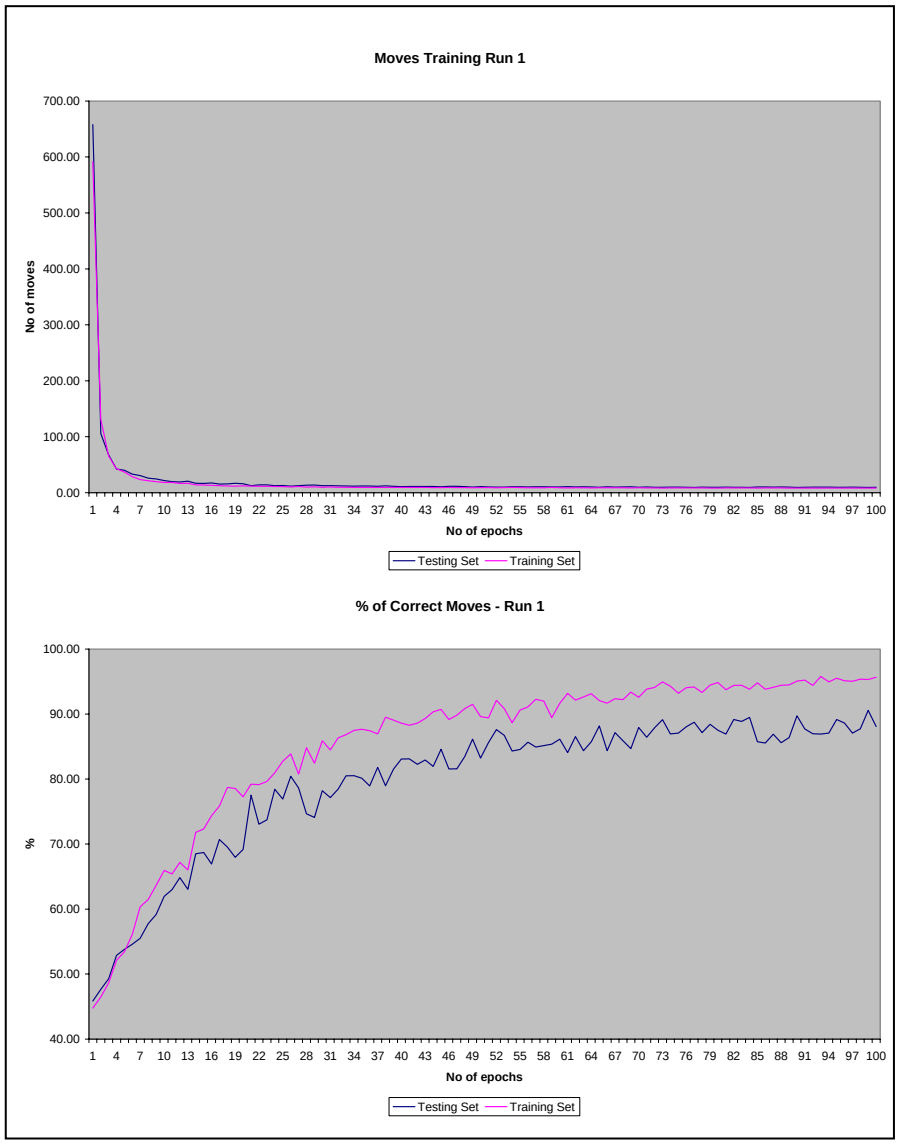

Figure 10 - Training Stats (Top) average number of steps required to reach the goal location during the testing of the network. (Bottom) percentage of correct moves made during the testing of the network.

Figure 10 shows the statistics gathered during the training of the network. After each epoch of training the network is tested both with the training data and the testing data. Here the samples are presented to the network and data gathered about *(i)* number of steps needed to reach the goal and *(ii)* the percentage of correct moves made by the network at each step. During this testing of the network training was prohibited and the relevant statistics gathered. This was done three times with all results being similar. Figure 10 (top) shows the average number of moves needed after each epoch for the goal to be reached. An average is taken for both the test and training set so the test value is averaged over the 500 test samples

and the training over the 1000 training samples. Initially, with no training, it takes on average approximately 650 steps for the agent to reach the goal location. This rapidly decreases and settles to about 10 steps after roughly 30 epochs, the number of steps required for the testing and training sets are very similar and the performance is as good on the testing set as the training set.

Figure 10 (bottom) illustrates the percentage of correct moves made at each step during the testing of the network. As expected initially, as the agent moves randomly the number of correct moves is roughly 50%, as there are two actions to be performed. This steadily rises during training, however, this doesn't stabilise after 30 epochs like the number of moves does. The performance keeps improving although the rate of improvement does decrease after approximately 60 epochs. In addition, the testing set doesn't perform as well as the training set does during the testing; this doesn't affect the average number of moves required to reach the goal.

5 Discussion

The developed system has been successful in allowing the robot to approach the table from random places in the environment. Once the orange docking is linked, the scenario will be complete. Reinforcement learning has been successfully used in two of the phases of this application. This illustrates that reinforcement learning is a viable option for use in robot navigation tasks.

This poses quite an interesting question; humans can easily see distinctive differences in tasks; would we be able to train a computer to do a similar thing? Instead of the programmer splitting the state space, could the computer automatically partition the state space? This has been approached in [18 & 19]. In these papers different techniques are adopted to partition the state space. Simple portioning of the state space would not have been a viable option in our approach as one network would be needed for the entire system. However, this would result in a large state space covering in our application the visual inputs of the omnidirectional camera as well as the pan-tilt camera. Therefore we have addressed this "curse of dimensionality" problem by segmenting the task into phases resulting in two smaller manageable state spaces.

Investigation could be made into improvements in the network to enhance the percentage of correct moves made. Some possibilities could include increasing the number of samples in the training set to increase the coverage of the training, allowing more starting locations to be trained. Another possibility could be to adjust the training algorithm to allow a smoother degrade in the strength of the critics weights. As the agent move away from the goal location there are large decreases in the strength of the weights to the extent that when the landmark appears behind the agent the critic's weights are very weak so the agent may still be moving randomly in this section. A smoother decrease in the critic's weights would allow this section of the network to have stronger weight connections and thus improve the performance of the network. There is one method that could be used to improve the network instantly which would be to switch from exploration of the environment to exploitation. Here the actor unit would be chosen which would give maximum reward. This however could lead to suboptimal solutions if used too early in training.

An alternative to the developed system could be to pan and tilt the camera that is supplied with the robot to find the target from a large distance and perform the

whole action based on this visual information. So instead of keeping the camera in a fixed position the camera could be moved to locate the table and object. This requires a coordinate transform to allow the calculation of the angle to the object given the odometry of the robot, the perceived position of the orange on the camera image and the pan of the camera. This is also an approach which we are currently pursuing [20] While such an approach enhances the range of an action strategy that relies on a single state space, there will remain situations in which a multi-step strategy has to be employed, such as if the target object is not visible from the starting point. Without the object visible, again one strategy is needed to get the robot close to the table and another for the docking to the object.

6 Summary

This paper has discussed the navigation system developed to allow the robot to firstly locate and dock to a table via a landmark. This greatly extended the range of docking of the system developed in [16]. Both systems (the original docking and the extended navigation) used the actor critic reinforcement technique to train the networks they used to achieve their goals. The extended navigation system trained its own network to allow the robot to move to the table, which has been demonstrated to work effectively. Once at the table the docking phase is able to complete the task. The navigation system developed has shown that reinforcement learning can successfully be applied to a real world robot navigation task. This system shows great potential for the development of a more advanced navigation system.

Acknowledgements

This is part of the MirrorBot project supported by the EU, FET-IST programme, grant IST-2001-35282, coordinated by Prof. Wermter.

References

1. Filliat, D. & Meyer, J.A. Map-based navigation in mobile robots. I. A review of localization strategies. J. of Cognitive Systems Research 2003, 4(4):243-282
2. Filliat, D. & Meyer, J.A. Map-based navigation in mobile robots. II. A review of map-learning and path-planning strategies. J. of Cognitive Systems Research 2003, 4(4):283-317
3. Dissanayake, M.W.M.G. Newman, P. Clark, S. Durrant-White, H.F. & Csorba, M. A solution to the simultaneous loacalization and map building (SLAM) problem. IEEE Transactions on Robotics and Automation 2001, 17(3):229-241
4. Tomatis, N. Nourbakhsh, I. & Siegwart, R. Hybrid simultaneous localization and map building; a natural integration of topological and metric. Robotics and Autonomous Systems 2003, 44:3-14
5. Guivant, J.E. Masson, F.R. & Nebot E.M. Simultaneous localization and map building using natural features of absolute information. Robotics and Autonomous Systems 2002, 40:79-90
6. Carelli, R. & Freire, E.O. Corridor navigation and wall-following stable control for sonar-based mobile robots. Robotics and Autonomous Systems 2003, 45:235-247

7. Maaref, H. & Barret, C. Sensor-based navigation of a mobile robot in an indoor environment. Robotics and Autonomous systems 2002, 38:1-18
8. Delgado, E. & Barreiro, A. Sonar-based robot navigation using nonlinear robust observers. Automatica 2003, 39:1195-1203
9. Menegatti, E. Zoccarato, M. Pagello, E. & Ishiguro, H. Image-based Monte Carlo localisation with omnidirectional images. Robotics and Autonomous Systems 2004, 48:17-30
10. Fiala, M. & Basu, A. Robot navigation using panoramic tracking. Pattern Recognition 2004, 37:2195-2215
11. Jogan, M. & Leonardis, A. Robust localisation using an omnidirectional appearance-based subspace model of environment. Robotics and Autonomous Systems 2003, 45:51-72
12. Gaussier, P. Joulain, C. Banquet, J.P. Lepretre, S. & Revel, A. The Visual Homing Problem: An Example of Robotics/Biology Cross Fertilization. Robotics and Autonomous Systems 2000, 30:155-180
13. Gaussier, P. Revel, A. Joulain, C. & Zrehen, S. Living in a Partially Structured Environment: How to Bypass the Limitations of Classical Reinforcement Techniques. Robotics and Autonomous Systems 1997, 20:225-250
14. Sutton, R.S. & Barto, A.G. Reinforcement Learning An Introduction. MIT Press 1998
15. Wörgötter, F. Actor-Critic models of animal control – a critique of reinforcement learning. Proceeding of Fourth International ICSC Symposium on Engineering of Intelligent Systems 2004
16. Weber, C. Wermter, S. & Zochios, A. Robot docking with neural vision and reinforcement. Knowledge Based Systems 2004, 12:165-72
17. Foster, D.J. Morris, R.G.N. & Dayan, P. A model of hippocampally dependent navigation, using the temporal learning rule. Hippocampus 2000, 10:1-16
18. Kondo, T. & Ito, K. A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robot control. Robotics and Autonomous Systems 2004, 46:11-124
19. Lee, I.S.K. & Lau, A.Y.K. Adaptive state space partitioning for reinforcement learning. Engineering Applications of Artificial Intelligence 2004, 17:577-588
20. Weber, C. Muse, D. Elshaw, M. & Wermter, S. Neural robot docking involving a camera-direction dependent visual-motor coordinate transformation. AI 2005 (submitted)