Cogn Comput (2009) 1:203–220
DOI 10.1007/s12559-009-9021-z

Actor-Critic Learning for Platform-Independent Robot Navigation

David Muse · Stefan Wermter

Received: 16 April 2009 / Accepted: 6 July 2009 / Published online: 24 July 2009

Abstract This article describes an approach in the field of reinforcement learning for robot control and a new Modular Actor-Critic architecture which supports platform-independent robot control. The architecture is tested on a landmark approaching task using movable pan/tilt cameras which successfully control both a large PeopleBot and a small Sony Aibo robot to perform the navigation task, with no retraining required. The architecture provides insight into the skills transfer between different robotic platforms and the modularisation of the architecture derived from splitting the control tasks into their component parts. The architecture and underlying principles could be used in rapid prototyping of new robotic platforms, where an already functioning control system can be used to allow more sophisticated navigation.

Keywords Reinforcement learning · Robot control · Robotics · Neural networks

Introduction

There have been different robot control systems developed over the years, from conventional, fixed-gain systems, to those that employ adaptive learning techniques [1–8]. However, so far Actor-Critic learning has received relatively little attention for robot navigation control, which is the focus of our approach in this article. The main purpose of robot navigation is to enable a robot to move around its environment, whether following a calculated or predefined path to reach a specific location or 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 [9, 10].

Many of the navigation systems implemented for robot navigation use manual coding, which leads to a lack of adaptability of the system although some systems have included learning (see [3, 4] for examples). Some training methods used for the learning systems have been various forms of reinforcement learning [11]. These learning algorithms overcome the problem of supervised-learning algorithms, as input/output pairs are not required for each stage of training but the only requirement is the assignment of a reward at the end—which can be a problem for multiple goals or complex systems [12]. However, for systems where there is just one goal, the reward will be administered only when the agent reaches the goal.

Busquets et al. [1] present a related approach, where a simulated robot is controlled using various reinforcement learning algorithms, coupled with the control architecture described by Sierra et al. [13] for the sharing of the visual component. Here, the vision system is used by different subsystems such as the navigation system to allow the robot to move to the landmark or the pilot system which performs obstacle avoidance. The reinforcement learning was used to learn how to share the visual component between the subsystems competing for its use. In contrast, Gaskett et al. [14] developed a control system for a robot using a fixed camera, and they used reinforcement learning to remove the need for camera calibration.

All of these cited systems have demonstrated the applicability of reinforcement learning to robot control problems. However, the main problem faced by reinforcement

D. Muse · S. Wermter (✉)
Department of Computing, Engineering and Technology,
Centre for Hybrid Intelligent Systems, University of Sunderland,
St. Peters Way, Sunderland SR6 0DD, UK
e-mail: stefan.wermter@sunderland.ac.uk
URL: www.his.sunderland.ac.uk

learning is the 'curse of dimensionality': increasing the number of inputs substantially increases the dimensionality of the state space, which is used to model the input variables [15]. With a larger state space, the training time of the networks increases. This is due to the increased number of states to be searched to find the goal state which used to be an issue debated in symbolic AI [16]. This issue is addressed in this research via modularisation of the state space into subtasks, which allows us to reduce the overall state space into several smaller state spaces to model the environment.

Reinforcement learning has been successfully developed for different robot control tasks. It has been used for instance to control a simulated robot in the field of robot soccer [2]. Hafner and Riedmiller concentrate on the results obtained from the simulated environment. However, they also state that they are working on adapting the system to work on their omnidirectional robot. They highlight a common problem which is the length of time it takes to learn the defined tasks. Reinforcement learning provides a powerful and flexible learning regime, but complex tasks require a larger state space, resulting in an increase in the search space during learning and operation.

In our early study, we developed a visual system, which allowed a PeopleBot robot to pick an object from a table [17]. This system used visual input and reinforcement learning to move the robot in such a way as to position the object between its grippers. However, it had a limited range, as the object needed to be in the visual field of the camera. The camera was held in a fixed position, so was unable to track the object if it moved out of view. As an extension to this early system, a system was designed and implemented to allow the pan/tilt camera to move [5]. This design increased the range of the robot vision, as the camera could track the object. It focused on a network developed to perform a coordinate transform to deduce the angle and distance to the object in robot-centred coordinates. These values were used as the inputs to the reinforcement network. However, the system was not developed for a real robot since the use of the coordinate transform network introduces possible errors to the system and increases the complexity, which is addressed and avoided in the system described in this article as relative landmark information is used.

An alternative extension, using an omnidirectional camera, was also pursued [6]. This was successfully implemented on the PeopleBot and used a sequence of reinforcement subsystems to allow the robot to reach the object from a greatly increased range. Here, a second network was added to locate the object via omnidirectional vision and a specified landmark. Once at the landmark the object was visible in the pan/tilt camera. However, as the original network was used, the camera remained stationary and if the object moved out of sight, the system would still be unable to track it. To avoid a lengthy online training time, the network was initially trained on a simulator. This was then exported to the robot for the final training, which fine-tuned the network to control the motor system of the robot. This principle will be used in this article to develop a new control architecture for different robots. Thus, this article describes an approach to develop a new robot control architecture based on more platform-independent reinforcement learning. By removing robot-dependent data for the control system, robot-independent control can be achieved.

Background of Reinforcement Learning

Reinforcement learning is a learning methodology where an agent can learn from experience by interacting with its environment. The learning is driven by the reward received during the interaction with the environment. This form of learning differs from supervised and unsupervised learning as there is no immediate reward. Reward is only received after a successful learning sample, which is then fed back throughout the network [11, 18].

As the agent interacts and explores the environment, it receives feedback in the form of a reward signal. The goal of the learning is to maximise the reward received for performing a given task. This raises an interesting question in reinforcement learning, which is how to trade off the exploration of the state space and exploitation of the reward function, "*The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task*" [11]. If the agent repeatedly follows the path with the maximum reward a global maximum may not be discovered and the current local maximum is continually used.

Reinforcement learning has its roots both in the fields of computation and psychology. One early inspiration for reinforcement learning was the findings of Pavlov with his experiments on a dog [19] who looked at classical conditioning where a stimulus preceded a reward, in this case food. Initially the dog salivated on the presentation of the food, however, after several trials the dog learned to associate the stimulus with receiving the food. When this occurred the dog started to salivate on the presentation of the stimuli, as the dog had learned that the food would follow.

For reinforcement learning, improvements are needed for the modelling of the state space to prevent what Bellman termed the 'Curse of Dimensionality' [15]. This is caused by the dimensionality needed to model the state space. Here every input used to model one aspect of the environment requires its own dimension. For example, if

an environment is modelled by four inputs and each one requires ten different states to model the input, there will be 10^4 possible states making a total of 10,000 individual states. For complex systems, huge state spaces may be required to model the environment, which is the motivation for function approximation and hierarchical methods to try to address this problem.

An overview of hierarchical methods is provided by Barto and Mahadevan [20]. By breaking complex problems into simpler tasks, smaller state spaces can be used to model the environment. Then, a method is needed to combine the different subsystems used to solve the complex task in hierarchical reinforcement models. For instance, Stringer et al. [21] developed a hierarchical model for motor control, where lower level primitive components learned individual motor tasks and a higher level component learned the sequence to execute the different primitives to complete the overall task. Tham [22] developed a hierarchical architecture for controlling a two-linked manipulator arm, whereas Morimoto and Doya [23] developed a hierarchical system to allow a robot manipulator with three links to stand up, to learn the individual actions required for standing, and to combine these actions into the correct sequence.

The work in hierarchical reinforcement learning also led to the concept of skills transfer, where trained subsystems could potentially be transferred to a similar task. An example of this could be a robot learning to grasp an apple and then to use the skills learned to grasp a cup. These tasks are similar but require a slightly different technique. In humans when we learn to perform a task, we can use the skills acquired in similar tasks. Research was conducted into using this concept to try and transfer skills learned in one task to ease learning of a similar task [20]. Also Singh et al. [24] present work in the field of skills transfer and Konidaris and Barto [25] outline the development of skills transfer between different learning problems, using the Options Framework. Here they aim to learn the options taken by the agent in what they term an Agent-Space, which decouples the learning of the options from the distinct State-Space which the leaning algorithm is using for learning the current task. It is hoped that by using the Agent-Space the learned actions can be used to speed up related tasks which have different state spaces. They term it intrinsically motivated reinforcement learning where the agent 'chooses' what it should learn. The agent then uses these skills to transfer the learned skills to similar tasks. This concept will be addressed in the research described in this article. However, the skills learned will not be transferred between similar tasks, but the control system that acquires certain skills in performing a task will be transferred between different robotic platforms.

Motivation for the New Architecture

It has been a trend over the years to develop robot control systems for a specific robot. However, using this approach to port the control system between robots, a redevelopment or total redesign of the system is required. This makes it difficult to recreate experiments and makes benchmarking difficult. It was the aim of this research to develop a control architecture using the Actor-Critic reinforcement learning algorithm to overcome this limitation in robotics. By developing the new architecture with decoupled control units, the need for robot-specific information was removed, allowing portability between robotic platforms. In this research, a modular architecture was developed comprising of different control systems. These control systems perform different subproblems of the overall task. The individual control units are decoupled in such a way that they have no direct input to each other but the action which one control unit produces influences other control units via the environment of the agent.

Developing control software for robots is a very time consuming and difficult process when the system is hand-coded [26] since all possible input and action pairs need to be considered. Hand-coding is manually programming input/output pairs to produce the required action. An alternative to hand-coding is for the agent to learn it over time from experience via interaction with the environment, and Wolpert et al. [27] discuss motor learning and the different approaches that can be used (supervised, unsupervised and reinforcement learning) for the robot to learn the control from a neurologists perspective. Mitchell et al. [28] present work on simple and reliable control of a mobile robot using neural networks, whereas Walter [29] produced some of the original work on mobile robot control. He used simple neural networks to control a simple Turtle robot.

It is the aim of this article to describe the development of a control architecture trained by Actor-Critic reinforcement learning to control two very different but related robot architectures, a PeopleBot and Sony Aibo robot. The two chosen platforms have similarities in that they both have moveable pan–tilt cameras and navigational capabilities. With the use of reinforcement learning, the networks can be partially trained in a simulator for camera and robot control. These partially trained networks can then be used to control the robots. Further training could then be carried out on the robots to specialise the networks with the actual movement of the individual robots and cameras.

As the architecture is designed to allow platform independence, it is vital that it is tested on different robot platforms. The first robot that was used for testing was a PeopleBot robot shown in Fig. 1.

Fig. 1 PeopleBot robot

The PeopleBot is a relatively large wheel-based robot with the dimensions of 47 cm width, 50 cm length and 124 cm height. The robot is supplied with a Canon VC-C4 Camera which is roughly 100 cm above floor level. The resolution of the camera is 460 × 350 and it can pan 100° to the left and right, and can tilt 30° up and 90° down.

The second platform to be used is a Sony Aibo dog shown in Fig. 2. The Sony Aibo is a four-legged robot and is much smaller than the PeopleBot. It has a width of 152 mm, a height of 281 mm and length of 250 mm. The camera is a 100,000-pixel CMOS sensor positioned in the nose of the robot which produces an image of a resolution of 172 × 143. As the camera is located in the nose it is pan/tilted by the movement of the robot's head. It is capable of tilting 20° up, tilting 65° down, and panning 90° to the left and right (these angles are approximate values). These different robots were chosen since they have significantly different platforms. If the architecture is successful in controlling both robots, it will provide some validation that the new architecture supports platform independence.

Even though the two test platforms are completely different, the architecture makes use of the similarities that are present. This is due to the availability of a pan/tilt camera in both robots. Both robots will be at the landmark when the cameras have a 0 pan/tilt angle and the camera is pointing downward with the landmark in the centre of the camera, as highlighted in Fig. 3b, c. The need for a coordinate transform mechanism has been avoided by the use of this principle and enables the platform independence.

Fig. 2 Sony Aibo robotic dog

Figure 3a, c shows the camera alignment of the PeopleBot and the Sony Aibo with a landmark located to the front right. Figure 3b, d shows the camera alignments when the landmark is located at the base of each of the robots. From Fig. 3 it can be seen that the two robots have very similar camera alignments in these situations, and it is this similarity that will support the platform-independent control.

Modular Actor-Critic (MAC) Architecture

Introduction

Given the assumption of the landmark being lower than the camera on the robot, the relative location of the landmark can be inferred. The pan of the camera can be used as the heading of the landmark from the robot and the tilt angle can be used to infer the distance of the landmark from the robot. However, with no constraint on the location of the landmark in the camera image, these inferred values would vary in accuracy. If there was a constraint that the landmark needed to be in the centre of the camera image, then the inferred values would be accurate. With these assumptions and constraints, this would work with any robot with a pan/tilt camera.

Figure 4 provides an example of the camera position at two different stages in the process of approaching the landmark. Figure 4a shows the camera at a position indicating that the landmark is to the front right of the robot. The robot control unit then needs to move the robot in such a way as to position itself with the landmark at its base and results in the camera position shown in Fig. 4b. The goal of the robot control unit is to control the robot's movement so that the camera control unit tracks the landmark and produces the camera alignment shown in Fig. 4b. With the

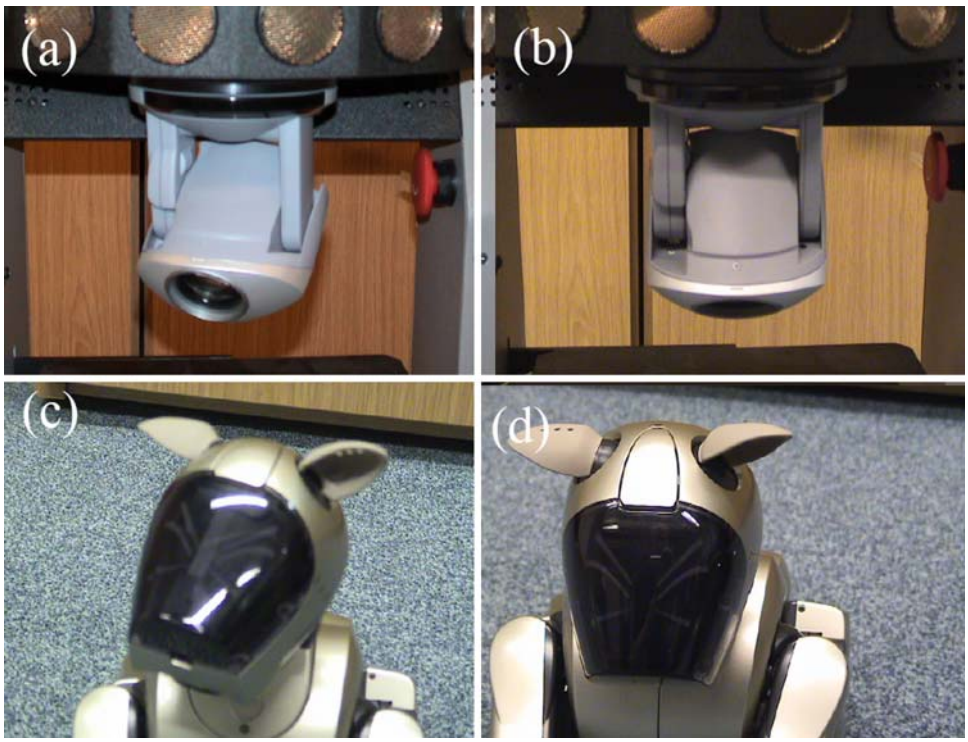


Fig. 3 Example of the camera alignments for the two test robots. **a**, **c** Cameras of the two robots pointing at a landmark to the front right of the robots. **b**, **d** Cameras of the two robots pointing at a landmark directly ahead and at the base of the robots

camera 'looking' down, the landmark is at the base of the robot. The robot control unit needs to learn the effect that the robot movement has on the relative position of the landmark and thus the movement of camera alignment. This should be learned in such a way that the robot can move to manipulate the alignment of the camera so that it is aligned as illustrated in Fig. 4b. In the example provided in Fig. 4, the robot would need to rotate to the right and move forward to produce the transition from Fig. 4a, b.

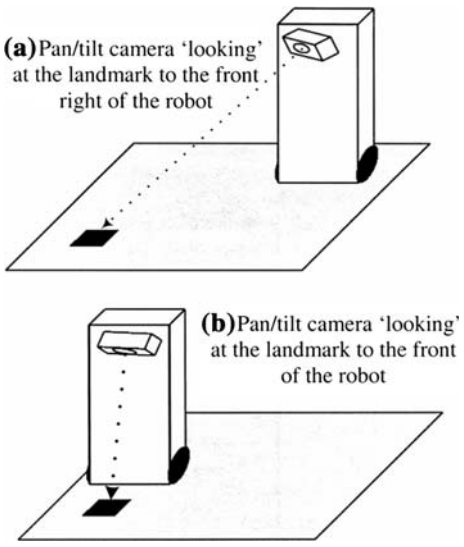
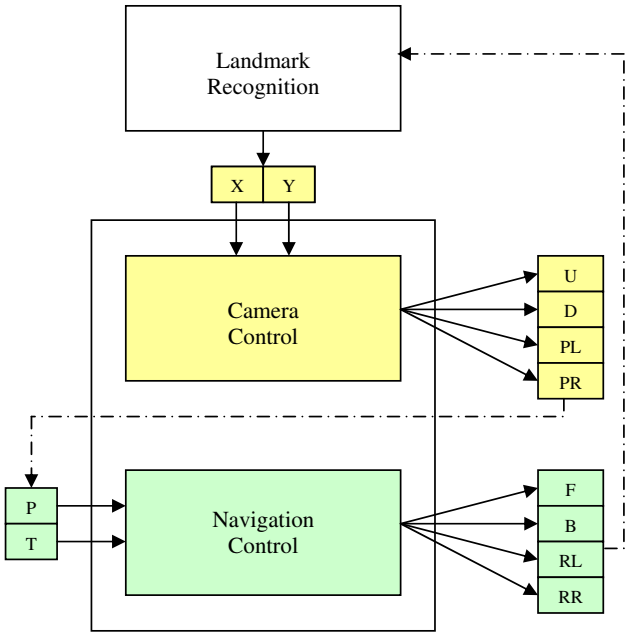


Fig. 4 Example of different camera alignments **a** position of the camera with the landmark located to the front right of the robot and **b** position of the camera when the robot is at the landmark

A control architecture can be designed and implemented to control robots with pan/tilt cameras and navigation capabilities. The core of the architecture consists of two control networks; one to control the pan and tilt of the camera and a second to control the motors which enable the robot to navigate. Two networks will be used since using one network would require a high dimensional hidden layer to model the state space and different input/output modalities. The input would be a combination of the location of the landmark and the camera alignment. The output of the network would have to combine camera and motor actions. We avoid this complexity with the modular design of the two networks. The visual recognition of the landmark will be decoupled from the core control allowing flexibility of the landmark to be approached. Figure 5 shows the proposed architecture.

In the testing scenario, there will initially be a direct line of sight from the robot to the landmark. However, the landmark may not be in the visual field of the robot's camera, requiring a search mechanism to be present in the overall architecture. This unit is to move the camera and robot until the landmark is present in the camera image. As this architecture is being developed to allow a more platform-independent control system, the landmark detection unit will be kept simple; it is not in the scope of this research to test advanced object recognition systems, but the portability of the proposed architectures between robotic platforms. Hence the landmark will be a unique object in the environment and will not be occluded by other objects.



Key:

X =	X coordinate of the landmark in the camera image
Y =	Y coordinate of the landmark in the camera image
U =	Tilts the camera upward
D =	Tilts the camera downward
PL =	Pans the camera left
PR =	Pans the camera right
P =	Pans angle of the camera
T =	Tilts angle of the camera
F =	Moves the robot forward
B =	Moves the robot backward
RL =	Rotates the robot left
RR =	Rotates the robot right

Fig. 5 Overview of the MAC architecture. The core components are the two control units, which support platform independency. Only the landmark recognition may need altering slightly for each platform

Once the landmark is detected, control will be passed to the two control units to move the robot to the required location. To allow the camera control unit to track the landmark as the robot moves, it will need updating with the position of the landmark in the camera image. This will be achieved via the landmark recognition unit shown in Fig. 5. The function of this unit is to recognise the landmark in the camera image and get the pixel values of the centre of the landmark to be passed to the camera control unit. As the camera and robot move, this unit will be invoked to produce the new coordinates of the landmark. With the landmark centred in the camera image, the following will occur as the robot moves: (i) if the robot rotates to the left, the landmark will move to the right of the image, causing the camera to pan to the right to track the landmark, (ii) if the robot rotates to the right, the landmark will move to the left of the image, causing the camera to pan to the left to track the landmark, (iii) if the robot moves forward, the landmark will move to the bottom of the image, causing the camera to tilt down to track the landmark and (iv) if the robot moves backward, the landmark will move to the top of the image, causing the camera to tilt up to track the landmark.

The two control units require coupling, which will allow them to work in unison. There are several options available for the coupling of the units. Two possible couplings are (i) to have both control units running concurrently, altering the alignment of the camera and the robots motors simultaneously. The main drawback of this option results in the camera control unit not being fast enough to keep track of the landmark, which could be lost from sight. This could be addressed during experimentation by reducing the speed of the drive motors, allowing more time for the camera to realign to the new position of the landmark; (ii) to only initiate the robot control unit once the landmark was within a threshold distance from the centre of the camera image. However, this approach may lead to a 'stop–start' system, with the robot moving slightly then stopping until the camera realigns itself with the landmark. Therefore, for the remainder of the article it is assumed that (i) will be used for the coupling of the control units.

The two control units are trained using the Actor-Critic reinforcement learning. Each network is partially trained in a simulator to learn the general control required for the camera and robot movements. These partially trained units can then be exported to the robotic platform to be controlled. As the units are only partially trained, further training will be useful on the robot to optimise the specific control. Having already learned the basic control for the landmark tracking and robot control in the simulator, the time required for the training on the robot is greatly reduced. This training thus fine-tunes the control units to work with the actual movements of the camera and robot.

Figure 6 illustrates the architecture for the camera control unit. There are two input units, one critic unit and four output units. As cameras on different robots may have different resolutions, the two inputs will be the normalised x and y coordinates of the landmark in the image. As the landmark will cover many pixels, the centre point of the landmark will be used as input to the camera control unit. The hidden area will in effect cover the image, and the node that contains the centre point of the landmark will be the node to produce the required camera action to move the landmark closer to the centre of the image. The hidden area is covered by Gaussian functions to find the node that encodes the landmark position. Here, the coordinates are fed into the hidden layer, and the node which produces the highest firing rate is the node encoding the landmark position.

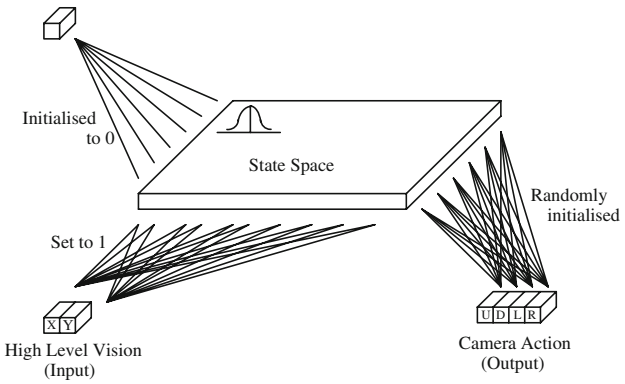


Fig. 6 Architecture used for the camera control unit from Fig. 5

The output of the network is the direction to move the camera and is one of the following four actions; (i) tilt upward, (ii) tilt downward, (iii) pan left or (iv) pan right. Once the camera has been moved, the new coordinates of the landmark are calculated and fed into the camera control unit, e.g. (11, 8). This will be repeated until the landmark is positioned in the centre of the image. This unit can be tested by moving the object through the camera image observing whether the camera is able to track the object.

Unlike the camera control unit, the robot control unit cannot work in isolation. This needs the camera control unit to track the landmark, as the robot moves through the environment towards the landmark. The architecture of the robot control unit is very similar to that of the camera and is shown in Fig. 7.

The architecture has two input nodes, one critic node and four output nodes. The hidden area encodes the input to the network of the pan and tilt angles of the camera. The network produces the required action to get the landmark to the base of the robot. The actions available are: (i) move forward, (ii) move backward, (iii) rotate to the left and (iv) rotate to the right. The move backward action may not be required, this will only be required if the robot is too close

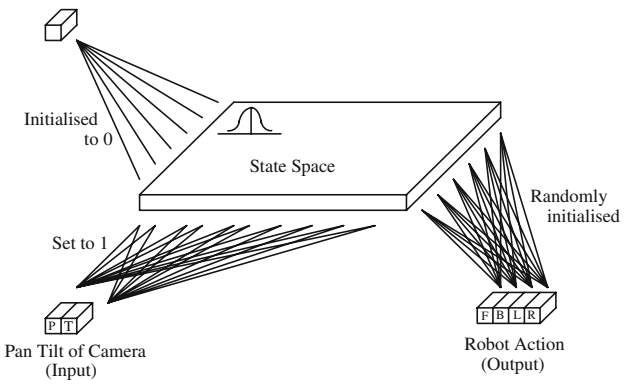


Fig. 7 Architecture used for the navigation control unit from Fig. 5

to the landmark and needs to retreat. The dimension of the hidden layer should be defined in such a way that the different pan/tilting capabilities of different robots can be represented. These angles will be normalised to feed into the network to produce the required motor action to move the robot towards the goal location.

Constraints Placed on the Control Architecture

It is the aim of the control architecture to be easily transferable between robotic platforms. Requiring as little re-coding as possible to port the architecture between the different robots, it is inevitable that there need to be slight differences in the code controlling the different robots, as they both have their own interface software to pass control commands to them. However, where possible, the code should be the same to generate the actions to be performed by the robots to get them to approach the landmark. The following list details the parts of the code that can differ between robots with explanations as to why these sections of code can change between robots:

- In the image processing, the colour of the predefined landmark may have a different colour value from the different cameras on the robots. The values for the colour threshold to segment the landmark from the rest of the image will be fine-tuned for the different cameras on the robots. However, the actual image processing used to segment the landmark from the image will be identical. The colour threshold values can be passed to the image processing on instantiation of the image processing class, allowing it to be passed as a variable at creation time. This means that the code performing the threshold will be kept constant between the different robots.
- The interface software for the two robots is different so both robots will have their own implementation of the main function, to allow the control architecture to interface with and control the different robots. The main function will initiate the link to the robot and include the control interface (Aria for the PeopleBot and URBI for the Aibo), allowing the control commands to be sent to the robots. The main function will also contain the instantiation of the control architecture and the image processing. As such, the main function acts as an interface between the control architecture and the robot.

Apart from the two points listed above, the rest of the code for the control architecture should be identical.

The developed system to be transferred between the two robotic platforms is tested in such a way so that they can not only be evaluated against each other, but they should also be evaluated against the system developed in the

exploratory experiments. This system was designed with a specific robot to complete the landmark approaching task. This evaluation will be able to show if the platform-independent system can perform as well as a control system which was designed for a specific robot.

All of the robots should be tested to check that they can complete the landmark approaching task successfully from a range of starting conditions, which should be kept consistent between the robots to enable direct comparisons of results. The closeness of the robot to the landmark, once the test has finished, can be recorded as a measure of success. As well as testing the complete system on the robots, the networks themselves which generate actions for the robot are tested and analysed in the simulated environment used for training. Also using the simulated environment will allow a more detailed analysis of the trained networks.

Training of the Control Networks

The Actor-Critic learning algorithm implemented for our architecture is based on the approach by Foster et al. [30]. Table 1 illustrates the learning routine of the Actor-Critic reinforcement learning algorithm.

Figure 8 illustrates how Eqs. 1–7 fit together in an overall network structure, based on Foster et al.'s work [30]. The figure shows the activations of all the cells from the hidden layer units, the critic and the actor units. It also shows the equations used to update the weights between the hidden layer and the actor units and the hidden layer and the critic. The probability function is shown, which is used in conjunction with a random number to generate the action taken by the agent. Finally, the equation to calculate the error of the produced action is included.

Table 1 The Actor-Critic reinforcement learning algorithm

- Pass inputs to network, activating the State Space
- Find the State Space unit with the highest activation
- Calculate the activation for the Critic and the Actor units
- Generate a random number between 0 and 1
- Check the activation of the first actor units
  - ○ While the activation is less than the random number produced
    - ■ Add the activation of the next actor unit to the sum of all activations checked
  - ○ The Actor units that caused the sum of all activations to pass the threshold of the random number is the action to be taken
- Perform the chosen action and calculate the activation of the Critic unit
- Calculate the update value given the two Critic values
- Update the Critic weights by adding the update value to the Critic weight connecting the original State Space unit to the Critic
- Update the weight of the Actor unit which generated the action taken by adding the update value to the current weight

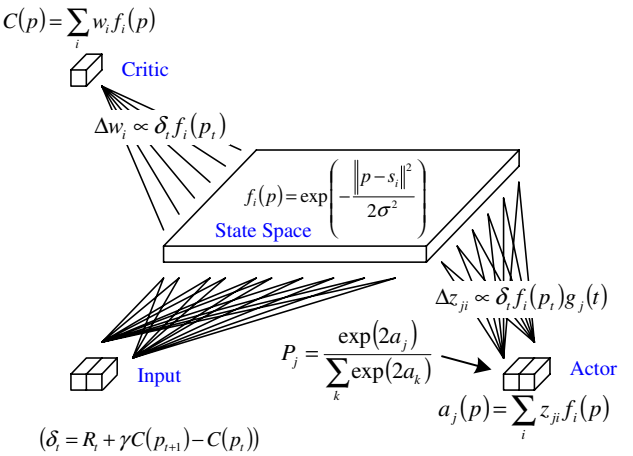


Fig. 8 Equations used by the Actor-Critic reinforcement learning algorithm

Equation 1 describes the firing rate of the hidden units. The firing rate is defined as

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

where p is the perceived position of the target, i is the index of the hidden unit, s_i is the centre point of neuron i and σ is the standard deviation of the Gaussian. The firing rate C of the Critic is calculated using Eq. 2 and has only one output neuron as shown in Fig. 3. The firing rate of the critic is thus a weighted sum of all the firing rates of the place cells, where w_i is the weight connecting the critic to the hidden unit at i as shown in Fig. 3.

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

To enable training of the weights of the critic, some method is needed to calculate the variation in rewards generated by the possible moves to be made by the agent. This is made possible by Eq. 3, based on the derivation of this equation in [30]. R_t in Eq. 3 is the reward given when the agent has reached the goal state. At all times this value is zero except when the agent has reached the goal. Here, R_t equals one as the reward is given when the goal has been achieved. $C(p)$ is the firing rate of the critic before a move was made and $C(p_{t+1})$ is the firing rate of the critic after the move has been made. γ is the constant discounting factor and relates to the learning rate in traditional training of neural networks.

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

However, as R_t only equals one when the agent is at the goal location and $C(p_{t+1})$ is zero when this occurs and R_t equals zero when $C(p_{t+1})$ is one, they are never included in the calculation at the same time. As previously stated R_t

only equals one when the agent has achieved the goal at this point. $C(p_{t+1})$ equals zero as the agent is at the goal and no more moves are made. With the predicted change in reward, the weights of the critic are updated proportionally to the product of the firing rate of the active place cell and the change in reward. Hence, the critic stores the discounted reward of the network moving away from the goal (Eq. 4).

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

This concludes the equations that were used for the hidden units and the critic. Finally, there are the equations used for the actor. The actor encodes the action to be taken by the agent and the number of cells in the actor corresponds to the number of actions that can be taken by the agent. 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 Eq. 5 where $a_j(p)$ is the activation of the actor unit at index j connected to the hidden unit at index i, and z_{ji} is the weight connecting the hidden unit and the actor unit.

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

A probability is used to judge the direction that the robot should move in, which is illustrated in Eq. 6. Here the probability that the agent 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 as the probability of the trained action being taken approaches 1. P_j is the probability that action j will be taken, a_j is the activation of that action, a_k is the activation of action k occurring with k ranging from 1 to the total number of actions possible.

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

The actor weights are trained using Eq. 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.

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

Simulated Environment for the Two Control Networks

The network was designed to learn the effects that moving the camera up, down, left and right has on the position of the landmark in its image. The simulator was an abstraction of this and gave the general coupling between actions of the robotic camera moving and effects on the location of the landmark within the camera image. The network learned the general strategy required to get the landmark to the centre of its image from any starting location.

Figure 9 illustrates the simplified effects that the different moves of the camera had on the location of the landmark in the image and these are as follows:

- Tilting the camera up moved the landmark down in the image.
- Tilting the camera down moved the landmark up in the image.
- Panning the camera left moved the landmark right in the image.
- Panning the camera right moved the landmark left in the image.

These can be simply modelled in the simulator by keeping track of the location of the landmark in the image. Here the location was encoded as the x and y coordinates of the hidden node that encoded the position of the landmark in the image, i.e. the node with the highest activation. Assuming that moving the camera moves the landmark one node in the network, the following would occur for each action:

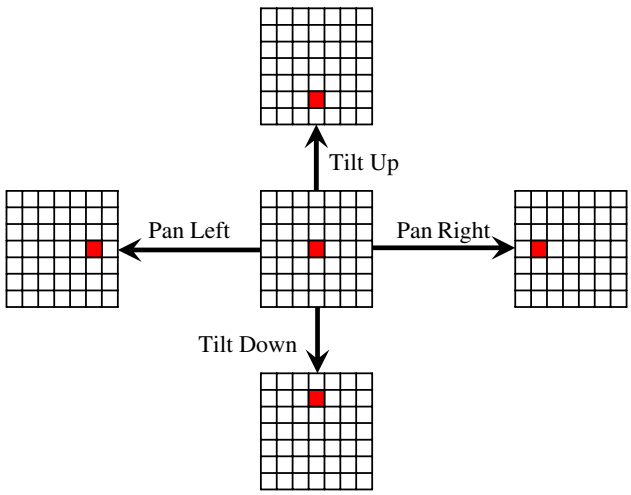


Fig. 9 Simplified effects of camera movement on the location of the landmark in the image. The *red square* represents the landmark. (Color figure online)

- Tilting the camera up added one to the y coordinate of the landmark.
- Tilting the camera down subtracted one from the y coordinate of the landmark.
- Panning the camera left added one to the x coordinate of the landmark.
- Panning the camera right subtracted one from the x coordinate of the landmark.

The origin of the network is in the top left corner of the hidden layer, as the origin of the images are in the top left corner. These effects have been simplified from what actually happened during movement of the real cameras, but was adequate to get the general strategy for moving the landmark to the goal location.

The location of the landmark in the image was normalised to the size of the network so it may take several moves of the real camera to get the position of the landmark in the camera image to move to the next node in the network. For example, with an image size of 640×480 and a network size of 16×12, each node covers a region of the image 40×40 pixels in size. If the movement of the camera at each step moves the landmark roughly ten pixels, it could take up to four steps to get the landmark to the next node in the network. This is another reason which makes it more appropriate for performing the initial training in the simulated environment, as it would only take one step to move from one node to the next. Once the general control is learnt by the network in the simulator, the fact that it may have to make the same move several times to move from one node to the next is less critical than during the initial training.

During the training of the network, the actions will be randomly chosen to allow exploration of the state space. As the training progresses, the actions chosen are expected to become more structured and the randomness of the exploration should decrease, as the agent is more likely to exploit paths with higher rewards, which would not initially be present. This should lead to a decrease in the number of steps required to achieve the goal. The goal set for the camera control network is to move the camera in the correct way to get the landmark to the centre of the image.

The random exploration is achieved using the probability Eq. 6 and the use of a random number between zero and one. As training progresses, the probability at each node that the appropriate action will be chosen increases and the movement of the camera should stop being random, and become more structured. During training, there is the possibility that an action is taken, resulting in the landmark moving out of view of the camera. This would occur if the landmark is perceived on the edge of the image and the action taken makes it move out of the view, i.e. if the detected landmark is on the far right edge of the network, the action to move the camera left would result in the landmark moving out of range. It was decided that if the action chosen lost the landmark from view, that action would not be taken and the weight of that action would be decreased by 0.1. This should suppress actions that would make the landmark go out of view. The network was trained using a simulator implemented in C++ using rules to encode the outcome of the movement of the robotic camera, as illustrated in Fig. 9.

The aim of the motor control network was to enable the agent move towards the landmark given the pan and tilt angles of the camera. The goal of the network was to produce a sequence of actions on the agent to move towards the landmark from the starting location. The four actions were:

- Move the agent forward
- Move the agent backward
- Rotating the agent left
- Rotating the agent right

At each time step all that were considered were current alignment of the camera. From the alignment of the camera, an action was produced to move the agent towards the landmark. The training was performed in a simulator. The effects of movement of the agent, with different relative locations of the landmark, were analysed. This allowed a simulator to be implemented for training. The results of the analysis are as shown in Fig. 10.

The diagram highlights the effect of moving the robot forward and backward on the pan orientation of the camera. It shows two different locations of the agent with landmarks to the left, right and straight ahead. The pan alignments of the camera at the different locations are also shown. From the diagram the following can be observed:

- With the landmark to the left, when the robot moved forward the camera needs to pan left to keep track of the landmark. The reverse is true when the robot moved backward.
- With the landmark to the right, the camera needs to pan right when the agent moved forward and left when the robot moved backward.
- From the above two points it can be induced that if the robot rotated to the left, the camera pans to the right and if the robot rotated to the right the camera pans to the left.
- Finally with the landmark directly ahead, the pan orientation of the camera did not alter as the robot moved forward and backward. However, the tilt orientation altered as the robot moved forward and backward, as shown in Fig. 11.

From Fig. 11, it can be seen that as the agent moved forward the camera tilted down to keep track of the

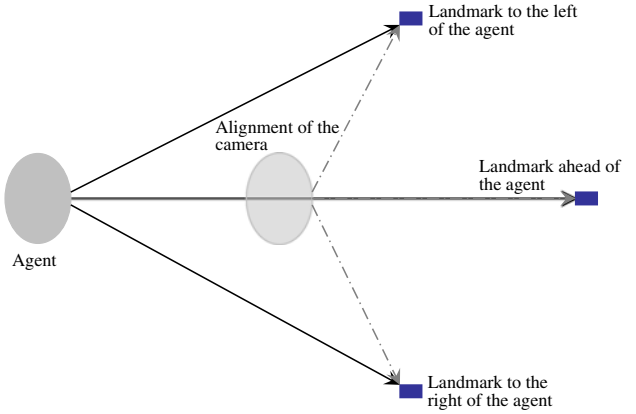


Fig. 10 Diagram to show the effects on the orientation of the robotic camera as the robot moves in a forward direction relatively to the landmark. It can be seen that if the landmark is to the left of the robot, moving forward causes the camera to pan to the right. If the landmark is to the right, moving the robot forward causes the camera to pan to the left. Finally, if the landmark is straight ahead the pan angle does not change

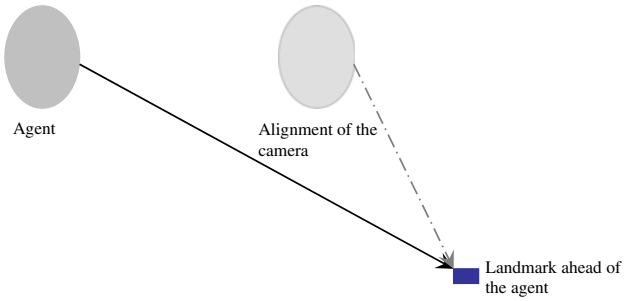


Fig. 11 Diagram showing the effects on the pan angle when moving the robot forward

Table 2 Rules incorporated into the simulator for the training of the navigation control network

Action	Relative location of the landmark		
	Left	Ahead	Right
Forward	Pan left	Tilt down	Pan right
Backward	Pan right	Tilt up	Pan left
Left	Pan right	Pan right	Pan right
Right	Pan left	Pan left	Pan left

landmark. The camera tilted up to keep track of the landmark when the agent moved backward. The rules that were incorporated into the simulator for learning of the movement control are listed in Table 2.

The rules were modelled in the simulator, to keep track of the orientation of the camera. Here, the orientation was recorded as the pan and tilt angle of the robotic camera, which had the landmark in the centre of its image. Movements of the robot changed the position of the landmark in the image, causing the camera to track the landmark.

Testing of the MAC Architecture

This section discusses the results collected during the testing of the camera control networks in the simulator. The hidden area sizes that were trained to perform the control task of tracking the landmark are:

- 5×5
- 9×7
- 10×10
- 17×13
- 20×20
- 33×25

Each network was trained with the following gamma values: (i) 0.6, (ii) 0.7 and (iii) 0.9. These values were chosen after initial empirical experiments, where we found that 0.5 was too low a gamma value and 0.99 was too high a value. Each network was trained with each of the gamma values five times, making a total of 90 trained networks. The summarised results are discussed below. Table 3 displays the best one performing of each of the different network size and gamma combinations, with Fig. 12 displaying the results graphically.

It can be seen in Fig. 12 that all networks had a similar performance level, with all errors being within 0.1 of each other. The best performing networks were used in testing the two different robotic platforms. The simulated environment was used to evaluate if the network could achieve the goal in the smallest number of moves. The testing on the two test platforms was conducted to evaluate how well the camera control networks performed the task of aligning the camera to the landmark. The performance measure for this testing was the number of pixels from the centre of the landmark, to the centre of the image.

Tests Conducted on the Motor Control Module of the MAC Architecture Using the Simulator

This section displays and discusses the results gained from testing the navigation control network. The errors from the

Table 3 Test results of the camera control network in the simulated environment

	Camera network					
Net size	5×5	9×7	10×10	17×13	20×20	33×25
Error	0.0769	0.0938	0.18	0.0991	0.125	0.0779

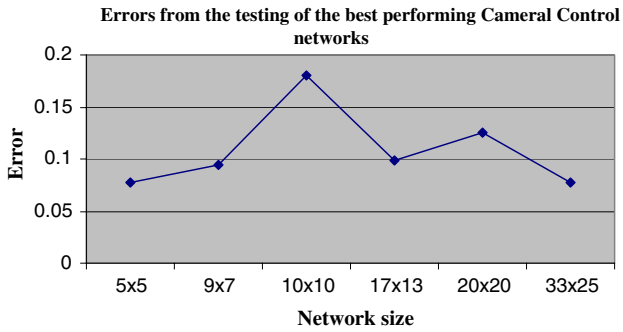


Fig. 12 Test results of the camera control networks in the simulated environment

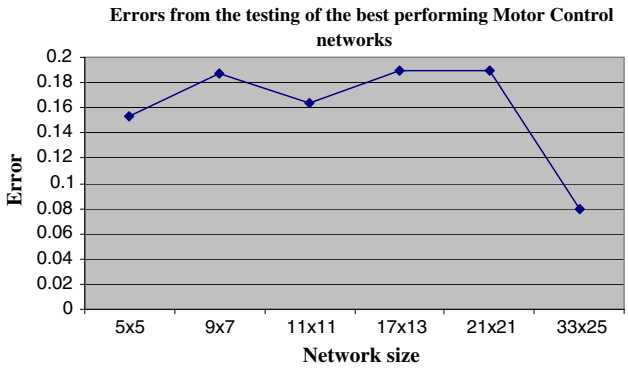


Fig. 13 Test results of the motor control network in the simulated environment

best performing networks are displayed in Table 4 and Fig. 13.

It can be seen in Table 4 and Fig. 13 that all networks trained in the simulated environment successfully learned to perform the navigation task. All networks produced performance levels within 0.12 of the other networks. During the testing of these networks in the simulated environment, no network produced results worse than making one wrong move for every five test samples presented.

As with the camera control network, it was expected that larger networks would perform the task more accurately. With the higher resolution of the state space, the larger networks can encode the state of the environment more accurately, allowing the larger networks to perform the task with a higher degree of precision.

The testing of both units of the MAC architecture has shown they were both able to complete the task they were trained to perform. Not only they could perform the relevant task, but they also performed it without making many wrong moves, all accomplishing the task in the simulated environment making less than one wrong move for every five samples. Hence, for over 80% of the samples presented, the networks were able to complete the task without taking a single wrong action. This was a high level of accuracy, considering one sample for a 33 × 25 network could require up to 26 actions to achieve the goal. However, the networks were able to accurately perform the navigation task of approaching the landmark. The following sections discuss the testing carried out on these networks for the two robotic test platforms.

Table 4 Test results of the motor control network in the simulated environment

	Motor network					
Net size	5 × 5	9 × 7	11 × 11	17 × 13	21 × 21	33 × 25
Error	0.1538	0.1875	0.1639	0.1892	0.19	0.0799

Tests Conducted on the Camera Control Module of the MAC Architecture Using the Two Test Platforms

This section presents the results gathered from the isolated testing of the camera control networks on the two different robotic platforms. Here, the landmark was placed in one of nine starting locations, shown in Table 5. However, due to the constraint of the landmark always being in sight, the camera was initially aligned with the landmark in view. The robot was required to align the camera with the landmark appearing in the centre of the image. There were two performance measures recorded during the testing, (i) the robot aligned the camera with the landmark in the centre of the encoded state space within 60 s. If this was not achieved the trial counted as failed. (ii) The second performance measure recorded was the distance in pixels from the centre of the landmark to the centre of the image.

Both test platforms had a maximum pan range which was less than 120°. The limit in pan range was why it was predicted that experiments 1 and 7 will fail. It was predicted that every other experiment would successfully align the camera with the landmark. The different size networks had different levels of accuracy with aligning the camera with the landmark. It was this accuracy that was investigated in these experiments.

Table 5 Experiments conducted on the camera control unit with the two test platforms

Experiment no.	Orientation (degrees)	Expected result
1	−120	Cannot centre landmark
2	−90	Centre landmark
3	−45	Centre landmark
4	0	Centre landmark
5	45	Centre landmark
6	90	Centre landmark
7	120	Cannot centre landmark

Results from the Testing of the Camera Control Network on the PeopleBot

Tables 6 and 7 provide examples of the results recorded for each experiment. Table 6 shows the results gathered during the testing of the 5 × 5 networks, with Table 7 showing the results recorded for the testing of the 20 × 20 network.

From Tables 6 and 7 for each trial the distance between the centre of the landmark and the centre of the image was recorded. As expected all trials where the landmark was placed at ±120° were unsuccessful as the cameras mounted on the PeopleBot did not have the pan range to allow the camera to align fully with the landmark. This was the same for all six network sizes.

All networks successfully aligned the camera with the landmark when the landmark was within the pan range of the robotic camera. As expected each network was able to perform the task with varying degrees of accuracy. Tables 6 and 7 show the results of all trials performed on the 5 × 5 and 20 × 20 networks, where it can be seen that the 20 × 20 network performed with a higher degree of accuracy than the 5 × 5 network. This was due to the higher resolution in encoding the state space with the larger network. Table 8 shows the summarised results from all experiments performed on the PeopleBot. These results are then presented graphically in Fig. 14.

Table 8 Summarised test results of the camera control network on the PeopleBot

	Camera network					
Net size	5 × 5	9 × 7	10 × 10	17 × 13	20 × 20	33 × 25
Error	35.1	23.7	29.3	23.8	17.9	22.4
S. error	30.1	16.6	19.2	16	6.5	14

The Dist row displays the average error of all tests conducted, both successful and unsuccessful. The next row, labelled "S. error" displays the average error of all the successful tests, leaving out all of the results from the unsuccessful trials, where the landmark was originally outside the pan range of the camera. The only tests that proved unsuccessful were the ones with the landmark at an angle of 120° from the robots, past the max pan capability of the robots. The errors from these tests were the number of pixels between the centre of the landmark and the centre of the image.

From the results presented in Fig. 14, we see the improvement in performance as the network size increases.

Table 6 Test results from the experiments on the 5 × 5 networks on the PeopleBot

Landmark	1		2		3		4		5		Avg
	Suc	Dist	Suc	Dist	Suc	Dist	Suc	Dist	Suc	Dist	
−120	N	49	N	45	N	50	N	46	N	47	47.4
−90	Y	29	Y	32	Y	31	Y	31	Y	35	31.6
−45	Y	27	Y	32	Y	32	Y	33	Y	31	31
0	Y	24	Y	33	Y	31	Y	21	Y	33	28.4
45	Y	31	Y	33	Y	26	Y	36	Y	31	31.4
90	Y	21	Y	26	Y	24	Y	36	Y	33	28
120	N	47	N	44	N	48	N	47	N	52	47.6
Average error from all trials											35.0571
Average error from successful trials											30.08

Suc denotes the success of the trial, Y if the landmark was aligned to the centre of the network, otherwise it was recorded as N and Dist denotes the distance of the centre pixel of the landmark to the centre pixel of the camera image

Table 7 Test results from the experiments on the 20 × 20 network on the PeopleBot

Landmark	1		2		3		4		5		Avg
	Suc	Dist	Suc	Dist	Suc	Dist	Suc	Dist	Suc	Dist	
−120	N	40	N	43	N	40	N	50	N	45	43.6
−90	Y	9	Y	13	Y	12	Y	12	Y	8	10.8
−45	Y	3	Y	4	Y	5	Y	4	Y	7	4.6
0	Y	3	Y	8	Y	7	Y	5	Y	6	5.8
45	Y	3	Y	6	Y	5	Y	4	Y	7	5
90	Y	6	Y	8	Y	7	Y	5	Y	5	6.2
120	N	50	N	46	N	55	N	47	N	49	49.4
Average error from all trials											17.9143
Average error from successful trials											6.48

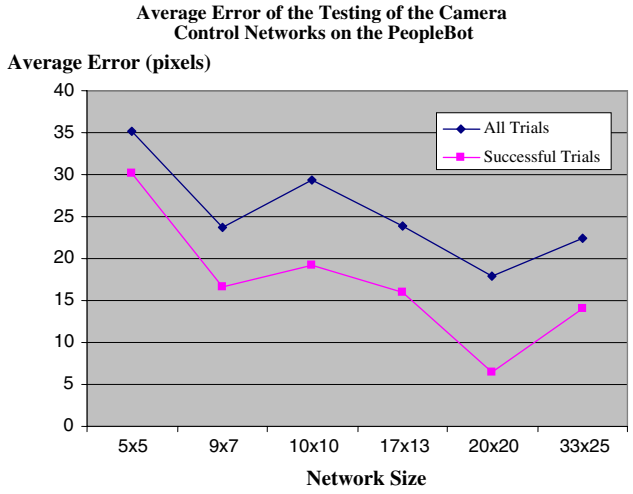


Fig. 14 Test results of the camera control network on the PeopleBot

The improvement can be seen for the two different networks, with the improvement increasing at a greater rate for the square networks. The 20 × 20 network was the best performing of all networks with an average error of only 6.5 pixels. These results demonstrate that the camera control unit of the MAC architecture has been successful in aligning the PeopleBot camera with the landmark.

Table 11 Test results of the camera control network on the Aibo

	Camera network					
Net size	5 × 5	9 × 7	10 × 10	17 × 13	20 × 20	33 × 25
Error	48.6	33.1	41.5	33.4	25.1	31.8
S. error	29.7	16.6	19.2	16	6.5	14.5

Results from the Testing of the Camera Control Network on the Sony Aibo

This section presents the results from the testing of camera control networks on the Sony Aibo. If the networks were successful in the task of aligning the robotic camera with the landmark, the fist module of the MAC architecture is successful in controlling two different robotic architectures to perform the same task. However, the success of the full MAC architecture is not verified until both control modules are tested in unison.

Tables 9 and 10 show the performance of the testing of the 5 × 5 and 20 × 20 camera control networks on the Sony Aibo. The performances of the networks were very similar to the results presented in Tables 6 and 7 of the testing of the networks on the PeopleBot.

Table 9 Test results from the experiments on the 5 × 5 network on the Sony Aibo

Landmark	1		2		3		4		5		Avg
	Suc	Dist	Suc	Dist	Suc	Dist	Suc	Dist	Suc	Dist	
−120	N	46	N	45	N	50	N	46	N	47	46.8
−90	Y	29	Y	32	Y	31	Y	31	Y	35	31.6
−45	Y	27	Y	32	Y	32	Y	33	Y	31	31
0	Y	24	Y	33	Y	31	Y	21	Y	33	28.4
45	Y	31	Y	33	Y	26	Y	36	Y	31	31.4
90	Y	21	Y	26	Y	24	Y	27	Y	33	26.2
120	N	47	N	44	N	48	N	47	N	51	47.4
Average error from all trials											48.56
Average error from successful trials											29.72

Suc denotes the success of the trial, Y if the landmark was aligned to the centre of the network, otherwise it was recorded as N and Dist denotes the distance of the centre pixel of the landmark to the centre pixel of the camera image

Table 10 Test results from the experiments on the 20 × 20 network on the Sony Aibo

Landmark	1		2		3		4		5		Avg
	Suc	Dist	Suc	Dist	Suc	Dist	Suc	Dist	Suc	Dist	
−120	N	40	N	43	N	40	N	50	N	45	43.6
−90	Y	9	Y	13	Y	12	Y	12	Y	8	10.8
−45	Y	3	Y	4	Y	5	Y	4	Y	7	4.6
0	Y	3	Y	8	Y	7	Y	5	Y	6	5.8
45	Y	3	Y	6	Y	5	Y	4	Y	7	5
90	Y	6	Y	8	Y	7	Y	5	Y	5	6.2
120	N	50	N	46	N	55	N	47	N	49	49.4
Average error from all trials											25.08
Average error from successful trials											6.48

Suc denotes the success of the trial, Y if the landmark was aligned to the centre of the network, otherwise it was recorded as N and Dist denotes the distance of the centre pixel of the landmark to the centre pixel of the camera image

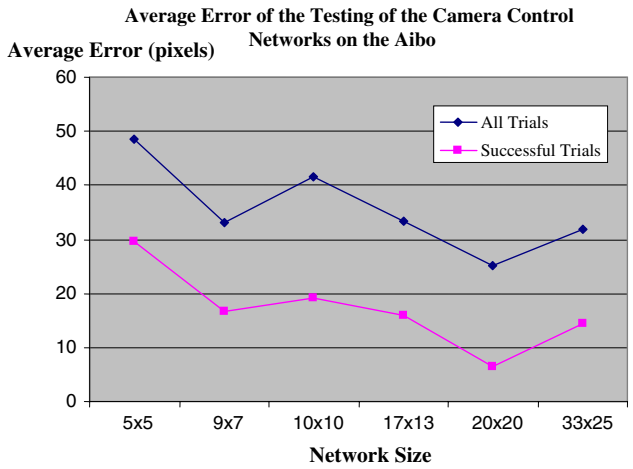


Fig. 15 Test results of the camera control network on the Aibo

The summarised results from the testing of the camera control networks are presented in Table 11 and graphically in Fig. 15.

The results gathered on the robots using the camera control network are different from some of the results gathered in the simulated environment. The testing performed on the robots was conducted to test the reliability of the networks to complete the tasks, whereas the testing in the simulated environment tested how quickly the network could perform the task. The best performing network was the 20 × 20 network, aligning the camera most accurately with the landmark. The results gained from the testing of networks on the Aibo and the PeopleBot were almost identical. This proved the camera control module was capable of controlling both robots to perform the same task.

Tests Conducted on the Full MAC Architecture

This section presents the summarised results gathered from testing the different combinations of networks on the different robots. Table 12 shows the tests that were conducted on the two test platforms.

The performance measures presented in this section for both robots were the distance in centimetres from the front of the robot to the landmark once the MAC architecture ran to completion. The error presented was the average error from all tests performed for the combination of camera and navigation control networks. Table 13 and Fig. 16 present the summarised results gathered on the PeopleBot with Table 14 and Fig. 17 presenting the same data gathered on the Aibo. Here, the results from all tests were averaged to get the performance measure.

The effect of the different combinations of control networks on the PeopleBot was apparent in Fig. 16. Here it can be seen that the navigation control network had the greatest influence on the performance of the robot completing the landmark approaching task. The performance of the MAC architecture was not influenced significantly by the size of the camera control network. There were only the tests conducted using the 5 × 5 motor control network which did not perform well, once the larger networks were tested the performance of the MAC architecture greatly improved. The larger two navigation control network sizes produced very good performance, allowing the robot to precisely approach the landmark. The camera control network did not have a great influence on the overall performance, since its task was to track the landmark by keeping it roughly in the centre of the image, allowing the motor control network to use the pan tilt angles of the camera to deduce the relative position of the landmark.

The testing of the MAC architecture on the Aibo can be seen graphically in Fig. 17. Again the size of the camera control network did not have a significant influence on the performance. This was for the same reasons given for the testing on the PeopleBot. The performance of the overall system did generally improve as the size of the navigation control network increased. The exception was a slight drop in performance with the 21 × 21 navigation control network. The overall performance of the MAC architecture was not as high on the Aibo. However, the MAC architecture proved it could still control the Aibo to move within an acceptable proximity of the landmark. The tests conducted on the PeopleBot saw all combinations of the testing with navigation control networks larger than the 5 × 5 producing results where the PeopleBot moved to within 5 cm of the landmark. This compares with the

Table 12 Test to be conducted with the two test platforms

Experiment no.	Orientation (degrees)	Expected result
1	−90	Rotate left and approach landmark
2	−45	Rotate left and approach landmark
3	0	Approach landmark
4	45	Rotate right and approach landmark
5	90	Rotate right and approach landmark

Table 13 Test results of the MAC architecture on the PeopleBot

		Camera network					
	Net size	5 × 5	9 × 7	10 × 10	17 × 13	20 × 20	33 × 25
Motor network	5 × 5	30.4	33.8	33.6	34.08	33.88	35
	9 × 7	2.64	2.76	2.72	2.52	3.04	2.36
	11 × 11	1.8	0.76	1.12	0.96	1.12	1.04
	17 × 13	0.08	0.08	0.08	0.04	0.08	0.04
	21 × 21	0	0	0	0	0	0
	33 × 25	0	0	0	0	0	0

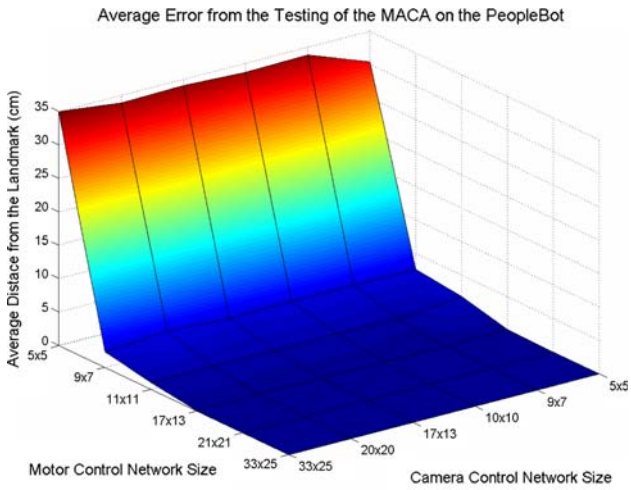


Fig. 16 Test results of the MAC architecture on the PeopleBot

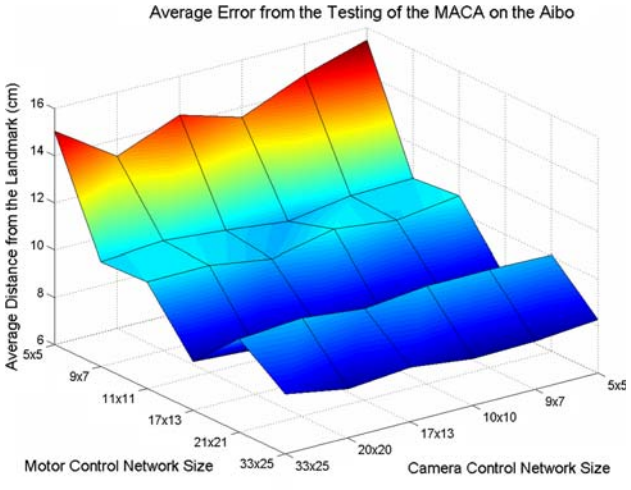


Fig. 17 Test results of the MAC architecture on the Aibo

performance of Aibo, where all networks larger than 5 × 5 enabled the Aibo to approach the landmark to within 11 cm.

The drop in performance on the Aibo was due to its movement, which was not as smooth while walking to the landmark. This walking motion caused a much less stable platform for the camera and navigation control networks. However, it has been demonstrated throughout these experiments that the MAC architecture can successfully control completely different robot architectures to perform the same task. The only code that needed altering was the low level interface of the control architecture with the two different robots.

Discussion and Conclusions

Overall, the aim of the article was to develop a control architecture based on the Actor-Critic learning algorithm, to control a PeopleBot and a Sony Aibo in the navigation task of landmark approaching. The development of the MAC architecture is related to and inspired by the research on hierarchical reinforcement learning [20, 23–25, 31]. With hierarchical reinforcement learning the problem to be learned is split into smaller tasks and the different units learning the smaller tasks are arranged in hierarchical architectures with low level control tasks being lower in the hierarchy than the higher-level decision making tasks.

Table 14 Test results of the MAC architecture on the Aibo

		Camera network					
	Net size	5 × 5	9 × 7	10 × 10	17 × 13	20 × 20	33 × 25
Motor network	5 × 5	15.48	14.68	13.56	14.36	13.28	15.04
	9 × 7	10.52	10.48	10.08	10.4	10.64	10.44
	11 × 11	10.72	10.36	10.68	10.12	10.52	10.52
	17 × 13	8.16	8.16	8.16	8.16	7.92	8.08
	21 × 21	10.08	10.16	10.16	9.88	10.16	9.96
	33 × 25	8.28	8.04	8	8.32	8.08	8.56

The architecture developed in this article is a modular architecture. Here the task of the landmark approaching scenario is split into its two fundamental task, (i) landmark tracking and (ii) navigation (landmark approaching). These two control units are independent of each other, with no direct coupling between the two; however, they are dynamically coupled as the output of one has an effect on the environment and subsequently the other control unit.

The performance varied depending on the network setup. The best performing networks from the simulated environment, which highlighted that they were the 'fastest' networks to conduct the task they were trained for, were tested on the two robotic test platforms. During the training the networks were tested, and the number of steps needed to perform the task was recorded as the performance measure.

The testing of the developed MAC architecture on the two different test platforms provided encouraging results, which highlighted the possibilities towards more platform-independent robot control, where a single architecture can take advantage of similarities between different robotic platforms, to be able to control them for the same task. These tests also highlighted the effectiveness of the Actor-Critic learning algorithm which was successful in training the two different modules in the MAC architecture to control the test platforms.

Our article has explored the possibilities of developing systems to bridge robotic platforms. It was the aim of this article to demonstrate that a central control system could be developed, taking into account similarities between different robotic platforms, to perform a set task. The task in this article was the learning of a landmark approaching task, and the testing on the MAC architecture has shown that control systems can be designed and learned to perform on multiple robotic platforms. By considering the similarities in the two different platforms, a generalised control system was developed, which successfully controlled the two robots in performing the same task.

The results have shown the possibilities towards more platform-independent control systems, allowing newly developed robots the possibility of having inbuilt control algorithms, which do not have to be designed specifically for only one robot. The use of such control systems have the potential to speed up the development of new robotic platforms, by removing the need for developing a new control system for the robot.

Acknowledgements Early stages of this study were supported partially by the MirrorBot project and NestCom projects coordinated by Prof. Wermter. Thanks go to Kim Forster for her constant support and encouragement, Dr. Kevin Burn for discussions and Chris Rowan who assisted in the setup of the robots and experiments.

References

1. Busquets D, Mantaras RL, Sierra C, Ditterich TG. Reinforcement learning for landmark-based robot navigation. Proceedings of the International Conference on Autonomous Agents and Multiagent Systems; 2002.
2. Hafner R, Riedmiller M. Reinforcement learning on an omni-directional mobile robot. IEEE/RSJ International Conference on Intelligent Robots and Systems for Human Security, Health, and Prosperity; 2003.
3. Kondo T, Ito K. A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robot control. Robot Auton Syst. 2004;46:11–124.
4. Lee ISK, Lau HYK. Adaptive state space partitioning for reinforcement learning. Eng Appl Artif Intell. 2004;17:577–88.
5. Weber C, Muse D, Elshaw M, Wermter S. A camera-direction dependent visual-motor coordinate transformation for a visually guided neural robot. Applications and Innovations in Intelligent Systems XIII—International Conference on Innovative Techniques and Applications of Artificial Intelligence; 2005. p. 151–64.
6. Weber C, Muse D, Wermter S. Robot docking based on omni-directional vision and reinforcement learning. Research and Development in Intelligent Systems XXII—International Conference on Innovative Techniques and Applications of Artificial Intelligence; 2005. p. 23–36.
7. Wermter S, Palm G, Elshaw M. Biomimetic neural learning for intelligent robots. New York: Springer; 2005.
8. Wermter S, Page M, Knowles M, Gallese V, Pulvermüller F, Taylor J. Multimodal communication in animals, humans and robots: an introduction to perspectives in brain-inspired informatics. Neural Netw. 2009;22:111–5.
9. Filliat D, Meyer JA. Map-based navigation in mobile robots. I. A review of localization strategies. J Cogn Syst Res. 2003; 4(4):243–82.
10. Filliat D, Meyer JA. Map-based navigation in mobile robots. II. A review of map-learning and path-planning strategies. J Cogn Syst Res. 2003;4(4):283–317.
11. Sutton RS, Barto AG. Reinforcement learning an introduction. Cambridge, MA: MIT Press; 1998.
12. 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.
13. Sierra C, Mantaras RL, Busquets D. Multiagent bidding bechanisms for robot qualitative navigation. Lect Notes Comput Sci. 2002;1986:198–205.
14. Gaskett C, Fletcher L, Zelinsky A. Reinforcement learning for visual servoing of a mobile robot. Proceedings of the Australian Conference on Robotics and Automation; 2000.
15. Bellman R. Adaptive control process: a guided tour. Princeton: Princeton University Press; 1961.
16. Lighthill J. Artificial intelligence: a general survey. Artificial Intelligence: A Paper Symposium. Science Research Council; 1973.
17. Weber C, Wermter S, Zochios A. Robot docking with neural vision and reinforcement. Knowl Based Syst. 2004;12(2–4):165–72.
18. Kaelbling LP, Littman ML, Moore AW. Reinforcement learning: a survey. J Artif Intell Res. 1996;4:237–85.
19. Pavlov IP. Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex; 1927. http://psychclassics.yorku.ca/Pavlov/.
20. Barto AG, Mahadevan S. Recent advances in hierarchical reinforcement learning. Discrete Event Dynamcal Systems: Theory Appl. 2003;13:341–79.
21. Stringer SM, Rolls ET, Taylor P. Learning movement sequences with a delayed reward signal in a hierarchical model of motor function. Neural Netw. 2007;20:172–81.

22. Tham CK. Reinforcement learning of multiple tasks using a hierarchical CMAC architecture. Robot Auton Syst. 1995; 15:247–74.
23. Morimoto J, Doya K. Acquisition of stand-up behaviour by a real robot using hierarchical reinforcement learning. Robot Auton Syst. 2001;36(1):37–51.
24. Singh S, Barto A, Chentanez N. Intrinsically motivated reinforcement learning. Proceedings of Neural Image Processing Systems Foundation; 2005.
25. Konidaris GD, Barto AG. Autonomous shaping: knowledge transfer in reinforcement learning. Proceedings of the Twenty-Third International Conference on Machine Learning; 2006. p. 489–96.
26. Smart WD, Kaelbling LP. Reinforcement learning for robot control. Proc SPIE: Mobile Robots XVI. 2001;4573:92–103.
27. Wolpert DM, Ghahramani Z, Flanagan JR. Perspectives and problems in motor learning. Trends Cogn Sci. 2001;5(11):487–94.
28. Mitchell RJ, Keating DA, Goodhew ICB, Bishop JM. Multiple neural network control of simple mobile robot. Proceedings of the 4th IEEE Mediterranean Symposium on New Directions in Control and Automation; 1996. p. 271–5.
29. Walter WG. A machine that learns. Sci Am. 1951;184(8):60–3.
30. Foster DJ, Morris RGN, Dayan P. A model of hippocampally dependent navigation, using the temporal learning rule. Hippocampus. 2000;10:1–16.
31. Singh SS, Tadic VB, Doucet A. A policy gradient method for semi-Markov decision processes with application to call admission control. Eur J Oper Res. 2007;178:808–18.