Learning Visual Docking for Non-Holonomic Autonomous Vehicles

Tomás Martínez-Marín and Tom Duckett

***Abstract*— This paper presents a new method of learning visual docking skills for non-holonomic vehicles by direct interaction with the environment. The method is based on a reinforcement algorithm, which speeds up Q-learning by applying memory-based sweeping and enforcing the "adjoining property", a filtering mechanism to only allow transitions between states that satisfy a fixed distance. The method overcomes some limitations of reinforcement learning techniques when they are employed in applications with continuous non-linear systems, such as car-like vehicles. In particular, a good approximation to the optimal behaviour is obtained by a small look-up table. The algorithm is tested within an image-based visual servoing framework on a docking task. The training time was less than 1 hour on the real vehicle. In experiments, we show the satisfactory performance of the algorithm.**

I. Introduction

Intelligent vehicles should exhibit an autonomous behaviour, learning from experience through interaction with the environment. Furthermore, they should learn on-line and update their internal dynamic models to maximize their short-term capabilities under changing conditions (e.g. terrain conditions, battery level, etc.). Most of the autonomous vehicles we can find in industry today follow a strict itinerary with a very limited interaction with the environment. In fact, the environment has been thoroughly adapted to the vehicles, evidencing the lack of intelligence in current systems.

In this paper, we present a generic approach for learning navigation of non-holonomic vehicles by visual interaction with the environment. The approach requires no calibration or geometric models, in contrast to conventional analytical solutions. Furthermore, the learned behaviour is robust to perturbations and noise.

The new algorithm is tested on a docking task for a non-holonomic autonomous vehicle, in which the car has to move towards a landmark located in the environment, docking just in front the landmark at a particular position and orientation (Fig. 7). In order to perceive the environment only an image sensor is used in this work, so we assume that there are no obstacles between the landmark and the vehicle, which can be easily detectable and avoidable employing range sensors (i.e. the laser mounted at the front of the vehicle).

The proposed algorithm speeds up the conventional Q-learning technique by applying memory-based sweeping [1] and reducing the number of allowed state transitions by enforcing the "adjoining property" [2], [3], a technique from the field of optimal control. This technique exploits the fact that in robotic applications, continuous sensory variables (interval numbers) are quantized into discrete states, where the natural ordering of states is preserved (ordinal numbers). While standard Q-learning assumes unordered states (nominal numbers), the adjoining property assumes that the states are ordered and allows only transitions between neighbouring states.

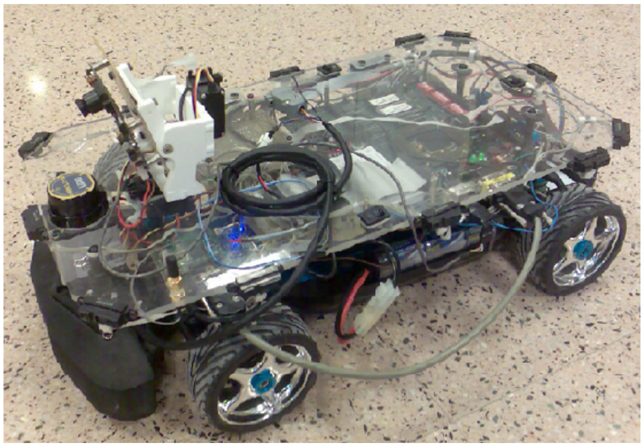

Fig. 1. The autonomous vehicle employed for the experiments.

Visual servoing is traditionally decomposed into position-based VS, which operates in the 3D Cartesian reference frame, and image-based VS, which operates in the 2D image plane [4], [5], [6]. An image-based visual servoing method is used, where the control law is computed directly from visual features, without explicit pose estimation.

In our approach, a separate tracking behaviour is used to control the vehicle's pan camera, in order to keep the object in the centre of the image at all times while the vehicle is moving. A minimal number of state variables are extracted from the image (concerning the apparent slope of the object edge) and the position of the camera (concerning the pan angle). These state variables are then used in the input to the motor controller of the autonomous vehicle.

In previous work [7], we tested our approach for learning forward motion on an Activmedia Peoplebot robot in a docking scenario. Here we extend this approach to a non-holonomic car-like vehicle, generalizing the learning with different controllers for forward and backward motion.

The rest of this paper is organized as follows. After a brief review of related work, Section II describes the vehicle platform. In section III we present some basic concepts of reinforcement learning. Section IV provides a brief introduction to the applied Cell Mapping techniques. Implementational aspects are addressed in section V and tested in Sections VI and VII through simulation and experimentation on a real

This work was supported in part by the Generalitat Valenciana under Project GV07/214, and by the University of Alicante under Project GRE/84

T. Martinez-Marin is with the Department of Physics, System Eng. and Signal Theory, University of Alicante, Spain

T. Duckett is with the Department of Computing and Informatics, University of Lincoln, UK

vehicle. Conclusions and suggestions for future work appear in section VIII.

A. Related Work

In principle, a robot could learn any task from scratch by reinforcement learning (RL) given enough time [8]. In practice, however, this time is too high for most complex tasks, and the designer must find some method to incorporate prior knowledge into the learning process.

Smart and Kaelbling addressed the practical issues of getting Q-learning to work on a real mobile robot [9]. The tasks investigated were wall following and obstacle avoidance. Learning was carried out in two phases: first, with the control policy being provided by a pre-programmed controller or a human with a joystick, and second, using the learned policy of the robot while RL continues ("fine tuning"). The tasks were also simplified by controlling only a single variable, the rotational velocity of the robot, while the translational velocity was controlled by a hand-coded algorithm.

Gaskett et al. [6] introduced a RL-based approach for training a mobile robot to wander (obstacle avoidance) and pursue a target using real-time vision. This was implemented in a subsumption architecture, such that target pursuit takes over from wandering when a valid target is detected. They used Advantage Learning to improve the optimal behaviour of Q-learning and a neural network to map the states to actions. Target pursuit was realised by visual servoing. This is simpler than our robot docking task because it requires servoing to a position but not to a particular orientation.

Weber et al. [5] used a neural network based approach to solve the docking problem on a holonomic robot by reinforcement learning. The learning was done in simulation, limiting the generality of the approach to tasks and environments that can be simulated accurately. In addition, in this work the pan-tilt mechanism was not exploited, which means that their visual controller is only valid when the robot is near to the object (40–50 cm). Our controller is valid for much longer distances of 4–5 m, only limited by the camera resolution.

In our previous work [7], we tested our approach for learning forward motion on an Activmedia Peoplebot robot in a docking scenario. In experiments we compared the new reinforcement learning algorithm with Q-learning, and also with a scheme using the linear controller as a bias to accelerate reinforcement learning. By analysis of the controllability and docking time, we found that the biased learning system improved on the performance of the linear controller, while requiring much less training than unbiased learning (less than 1 hour on the real vehicle).

II. VEHICLE MODEL

The reinforcement learning controllers were implemented on the nonholonomic vehicle shown in Fig. 1. The vehicle is equipped with an array of infrared sensors, laser and a pan-tilt camera (see Fig. 7). The vehicle is autonomous, using a microcontroller MPC555 to process all sensors and the RL controllers. In this work, only the vision sensor was used to estimate the state variables. We used a low resolution CMOS sensor (CMUcam2+) mounted on a turret. The range of the pan and tilt angles are ± 100 degrees and ± 25 degrees, respectively.

In order to simulate the car-like motion we will employ a simple model, although this model will not be used by the algorithm to find the optimal system behaviour. The state space formulation [10] of the vehicle model we will use is the following:

$$\begin{aligned} \dot{x} &= v_T \cos\theta \cos\varsigma, & (1)\\ \dot{y} &= v_T \sin\theta \cos\varsigma, & (2)\\ \dot{\theta} &= v_T \sin\varsigma. & (3) \end{aligned}$$

where v_T is the translational velocity and ς is the steering angle of the vehicle.

The distance between the reference point (x, y) and the middle point of the driving wheels is 0.32 m. The orientation of the car is denoted by θ. The two controls of a car are the velocity v_T of the driving wheels and the steering angle ς. It is important to note that the state equations are only used in the simulations to describe the robot trajectories. In the experiments, the model is built on-line by recording the transitions between states and the immediate rewards.

A. The state space variables

In many vision-based manipulation applications, information has to be extracted from the image concerning the position, orientation, size and shape of the object. In our case, it is not necessary to calculate a 3D pose estimate, it is sufficient to use a 2D pose estimate to correct the pan-tilt angles in order to keep the mark in the centre of the image. Instead of using the 2D position (relative x, y values in the image) as state variables, we use the pan angle of the camera and the orientation of the mark with respect to the vehicle.

Although the state space of the system is three-dimensional, the visual docking behaviour can be specified in a two-dimensional state space. In this case, the problem is simplified by fixing the value of v_T. Then, the task is reduced to controlling the steering angle of the vehicle (ς) in a two-dimensional state space (β, α), where α is the pan angle of the camera and β is the relative orientation of the vehicle with respect to the object. For the docking task, the goal state in this relative coordinate system is the origin. In our reinforcement learning experiments, we allow only three possible actions in each state (-20, 0, 20 deg).

B. Estimating the state variables on the real vehicle

The state variables (β, α) can be estimated through the variables (m, pan) respectively, which can be obtained from the image sensor. The variable m corresponds to the slope of the bottom edge of the mark in the image (see Fig. 2). The pan angle is valid if the mark is kept in the centre of the image. For reinforcement learning on the real robot, β is estimated using $\arctan(m)$, and α is estimated using the

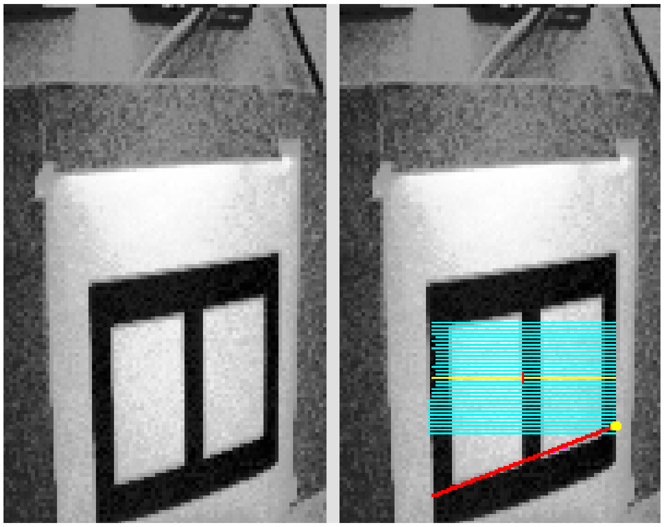

Fig. 2. Left: raw image. Right: visualisation of the image processing (87×143 pixels). The yellow line with the red point determines the centre of the landmark. The red line indicates the orientation of the landmark with respect to the vehicle.

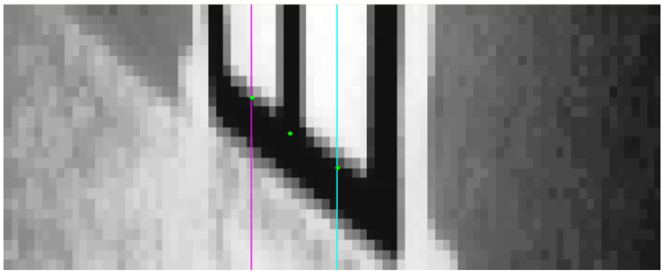

Fig. 3. Centred window of 87×30 pixels. The green points are used to estimate the orientation of the landmark with respect to the vehicle.

pan angle. The continuous state variables are converted into discrete states (cells) by uniform quantization (see section IV).

For the success of the docking behaviour, it is essential to track the landmark at all times while the vehicle is moving. For both the pan and tilt angles, Proportional-Derivative (PD) controllers are employed in order to keep the mark in the centre of the image [11]. The parameters of the controller were adjusted to avoid overshoot if the object or the vehicle changes its position suddenly.

The image processing employs two levels of image resolution, which are shown in Figs. 2 and 3. When the docking task starts the vehicle searches for the landmark, scanning the environment by turning the turret. In this stage, it acquires full images with the standard resolution (87×143 pixels). After image equalization, Fig. 2 shows the landmark detection by matching the image with a pattern (cyan lines) and then extracting the median (yellow line). The landmark orientation is obtained applying a line fitting algorithm (red line). When the landmark is situated at the centre of the image, the image resolution is reduced by selecting a window of 87×30 pixels, as shown in Fig. 3. In this way, the frame rate is increased to 5 fps. In this case, the image processing detects the centre of the bottom line and its slope (green point).

III. Reinforcement Learning

Reinforcement learning methods only require a scalar reward (or punishment) to learn to map situations (states) to actions [1]. As opposed to supervised learning, they do not require a teacher to acquire the optimal behaviour, they only need to interact with the environment learning from experience. The knowledge is saved in a look-up table that contains an estimation of the accumulated reward to reach the goal from each situation or state. The objective is to find the actions (policy $a = \pi(s)$) that maximize the accumulated reward in each state. Q-learning is one of the most popular reinforcement learning methods, since with a simple formulation it can address model-free optimization problems. The accumulated reward for each state-action pair $Q(s,a)$ is updated by the one-step equation

$$\Delta Q(s,a) = \alpha \left(r + \gamma max_{a'} Q(s',a') - Q(s,a) \right) \tag{4}$$

where Q is the expected value of performing action a in state s, r is the reward, α is a learning rate which controls convergence and γ is the discount factor. The discount factor makes rewards earned earlier more valuable than those received later. If the reward function is proper [12], the discount factor can be omitted ($\gamma = 1$). The action a with highest Q value at state s is the best policy up to instant t, which approximates the optimal behaviour when $t \rightarrow 0$:

$$a^* = \pi^*(s) = arg\ max_{a'} Q(s,a') \tag{5}$$

In applications with real systems Q-learning can spend a very long time making hundred of thousands of trials to approximate the optimal behaviour. To speed up the learning it is necessary to incorporate some planning. Prioritized sweeping [13] and Dyna-Q include a search procedure to simulate past real experiences in a specified order. In our simulations we will combine Q-learning with the search mechanism proposed in the new RL algorithm.

IV. Adjoining Cell Mapping

Q-learning was conceived for discrete state and action spaces, where the state space is not necessarily metric. In robotic applications the state space is continuous, so it is mandatory to discretize the state space into cells. The inherent discretization errors can produce a poor approximation to the optimal behaviour in complex nonlinear systems such as mobile robots. Cell Mapping techniques were conceived in order to deal with the discretization problems in an efficient way.

Cell-to-cell mapping methods are based on a discretization of the state variables of the system, defining a partition of the state space into cells [14]. A cell-to-cell mapping can be derived from the dynamic evolution of the system. In [14] a cell mapping application for the design of optimal controllers

```
Initialize $Q(s,a)$ and $Model(s,a)$
    $x \leftarrow$ current state
    $s \leftarrow cell(x)$
    IF s = sink or goal
    THEN $reverse(x)$
    ELSE $a \leftarrow policy(s)$
        Execute action $a$
        Observe resultant state $x'$ and reward $r$
        IF $D_k$-adjoining$(x, x')$
        THEN $Model(s,a) \leftarrow x', r$
            FOR all $(s,a)$, repeat N times:
                $\bar{x}', \bar{r} \leftarrow Model(s,a)$
                $s' \leftarrow cell(\bar{x}')$
                Update $Q$ table using Eqn. 4
UNTIL training terminated
```

Fig. 4. Reinforcement Learning Algorithm.

State variables: 2. (21 × 21 cells)	x_1: -80 $\leq \beta \leq$ 80°. x_2: -100 $\leq \alpha \leq$ 100°.
Objective state:	$(\alpha, \beta) = (0°, 0°)$
Control variables: 1. (3 actions)	(u_1: -20 $\leq \varsigma \leq$ 20°.)
Sampling time:	T_s: 0.1 sec.
Reward:	r = 100 if goal r = -20 if sink r = -n (T_s) otherwise
Adjoining distance:	D-2

TABLE I

PARAMETERS FOR THE RL ALGORITHM.

is proposed. This method (CSCM), based on the Simple Cell Mapping (SCM) technique, carries out a discretization of both state and control variables and uses a cost function to specify the desired optimality criterion. In [2], [3] the CACM algorithm for optimal control of highly nonlinear systems is proposed. This method is based on the Adjoining Cell Mapping (ACM) technique, whose central concept is the creation of a cell mapping where only transitions between adjoining cells are allowed [15].

The adjoining property states that the distance D_k between the current cell and the previous cell is equal to some integer value k equal or greater than 1. For our RL controller, we will define the adjoining property in terms of the continuous states $\mathbf{x}$ and $\mathbf{x}'$ as follows:

$$D_k(\mathbf{x}, \mathbf{x}') = \max_j |\frac{x_j - x'_j}{h_j}| = k, \tag{6}$$

where x_j indicates the j-th component of the state $\mathbf{x}$ and h_j is the cell size of the j-th dimension.

In Q-learning the transitions between states are evaluated at fixed sample times, while with our RL controller the transitions have to satisfy the adjoining distance condition in order to be evaluated. By appropriate selection of this distance with respect to the number of cells, it is possible to minimize quantization effects and better approximate the optimal behaviour of the system.

V. THE RL AGORITHM

The RL algorithm that implements the concepts described above is presented in Fig. 4. The state is represented in the algorithm by a real valued vector x, which is converted to the discrete state s (integer index) by the function $cell()$. In our experiments, uniform discretization was used with 21 cells per variable in simulation and 15 cells on the real vehicle (see Table I for full details of the RL parameters). The function $Dk - adjoining()$ is used to determine whether the adjoining property has been satisfied. The index s is used to update the Q-table, and x is used to update the function $model()$. Since the controller uses noisy data from an image sensor, the function $model()$ estimates the state of the system by filtering before storing it. In our experiments an average filter was used.

For the docking behaviour the aim of the controller is to move the vehicle from any initial position inside the region of interest to the object position through a minimum-time trajectory. A trial finishes when the vehicle moves outside of the state space (sink cell) or when it enters in the goal. Then, the function $reverse()$ moves the car backward, using its vision system to keep the object in the centre of the image, until some starting position inside the state space is reached. The function $policy()$ selects an action for each transition of the system. The RL controller selects the actions randomly to explore most of the state space during training. Other alternatives such as an ϵ-greedy or softmax exploration policy do not introduce significant benefits in this application, since they reduce the exploration of peripheral states, thus delaying the growth of the controllability region. In the update rule (Eqn. 4), the learning rate α is variable, falling inversely with the number of transitions and the discount factor is fixed to $\gamma = 1$.

The docking task is symmetric in the state and action spaces, i.e., the actions taken when the robot is to the right of the object are symmetric with respect to the actions on the left side. Each cell has a *mirror* cell that satisfies this property. For each transition, we exploit this symmetry by also updating the model and the Q-factor for the mirror cell.

VI. DOCKING BEHAVIOUR

The docking behaviour has been implemented using the RL algorithm described in the former section. Two improvements have been introduced to the algorithm:

First, in order to speed up the learning time the function $policy()$ combines random actions with the action provided by a simple linear controller that follows the equation: $V_R = K_b\beta + K_p pan$, where β and pan are the state variables measured on the vehicle. The parameter values used in our experiments were $K_b = 0.3$ and $Kp = 0.5$.

Second, the vehicle should move forward and backward to increase the region of controllability. For that reason, during

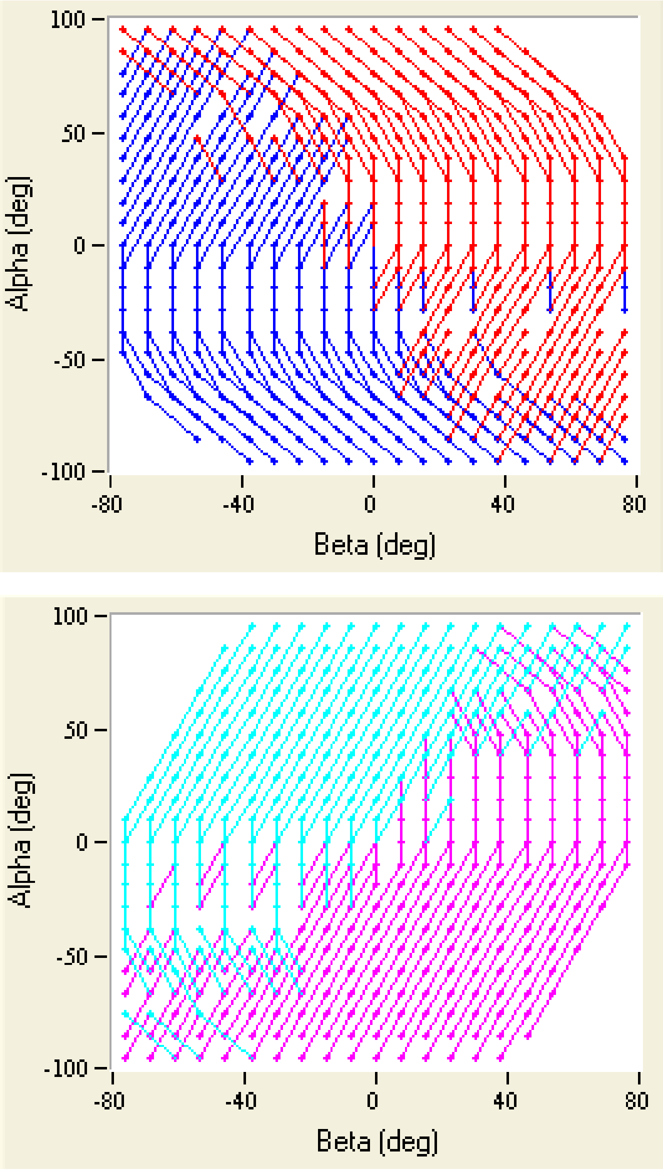

Fig. 5. State space (β,α) of the vehicle behaviour.Top: Forward motion. Bottom: Backward motion

the training phase two RL controllers are built: one for forward motion and the other for backward motion. Both controllers are built using the same function $model()$, so the learning time remains the same. These controllers could later be used by a three-dimensional (3D) RL controller including the state variable *distance*. This 3D controller would learn to select the optimal position where the 2D controllers (forward and backward motion) are switched. In this way, the path planning can be improved using a control architecture with two levels of abstraction. Fig. 5 shows the state space of both controllers. We can observe that the controllers approximate the time-optimal behaviour: bang-bang control with the classical *switching curve* in the centre. Finally, Fig. 6 shows some trajectories of the proposed controller. We can see that the vehicle is able to reach the goal using forward and backward motion.

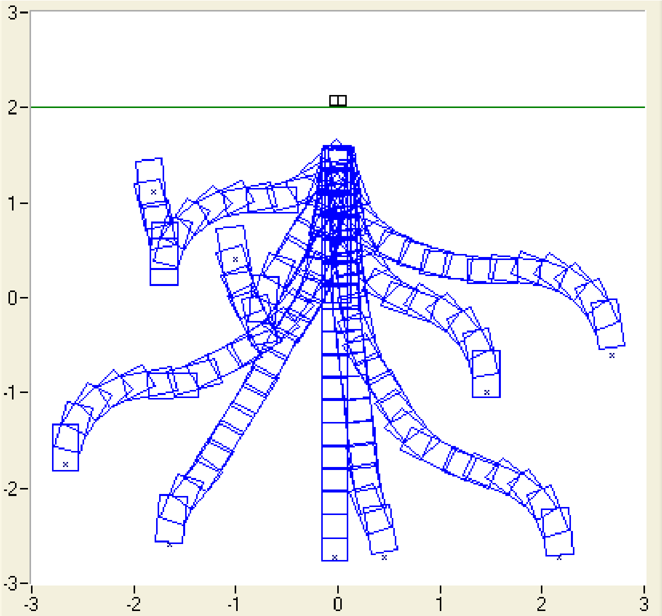

Fig. 6. Some trajectories of the vehicle with initial orientation $\theta = \frac{\pi}{2}$.

VII. Robotic Experiments

In this section we present the experiments carried out on the real vehicle described in section II. The first step was to train the vehicle in a square area of $2.8 \times 2.8\ m^2$ with landmarks at the four sides. Thus, the function $reverse()$ was simply to move the vehicle backwards, since there is always some landmark in the field of view. The training time to acquire a good approximation to the time-optimal behaviour for both forward and backward controllers was 60 minutes. Fig. 7 shows the docking behaviour on the real vehicle. In this trajectory, the car moves to the landmark turning to the left, but the steering angle (only 20 degrees) is not enough to reach the goal in a single trajectory. Thus, when the car reaches the minimum distance threshold (measured by the landmark size in pixels), the controller switches to backward motion. The aim of this controller is to centre the vehicle as it is going backwards. Finally, when the car goes beyond a second distance threshold, the controller switches again to forward motion reaching the goal. The use of both forward and backward controllers increases the system controllability, only limited by the pan angle. Furthermore, the visual docking can be done from a long distance (4–5 meters, only limited by the image resolution) since the controller does not have any distance restriction for forward motion. The experimental results show that the reinforcement learning controllers have been implemented successfully on the autonomous vehicle.

VIII. Conclusion

We presented a solution for vehicle docking using reinforcement learning in a visual servoing framework. A new RL

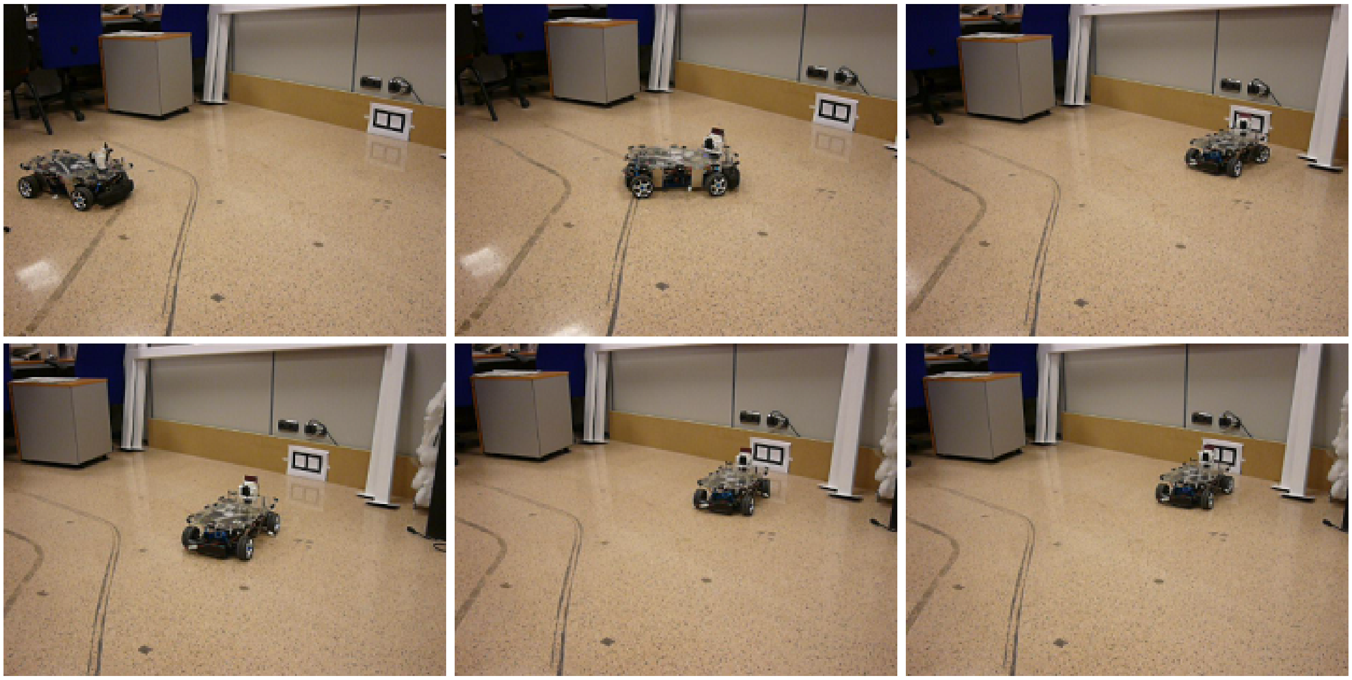

Fig. 7. Sequence of images showing the docking behaviour learned on the non-holonomic vehicle.

algorithm was presented that is better suited to real vehicles that operate in continuous state spaces (by exploiting the adjoining property and model-based sweeping). The approach requires no calibration or geometric models, and the reactive behaviour is robust to perturbations and noise. The closed loop solution is based on a relative coordinate system: no global reference frame is required, so the system is robust to positioning errors, e.g., due to odometry drift.

In future research, we intend to extend the approach to more complex tasks and robot vehicles with higher dimensional state and action spaces.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[2] P. Zufiria and T. Martínez-Marín, "Improved optimal control methods based upon the adjoining cell mapping technique," *Journal of Optimization Theory and Applications*, vol. 118, no. 3, pp. 657–680, 2003.

[3] T. Martínez-Marín, "Optimal path planning for car-like vehicles in the presence of obstacles," in *Proc. IEEE Int. Conf. on Intelligent Transportation Systems*, Shanghai, 2003, pp. 1161–1164.

[4] Y. Mezouar and F. Chaumette, "Path planning in image space for robust visual servoing," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, San Francisco, 2000, pp. 2759–2764.

[5] C. Weber, S. Wermter, and A. Zochios, "Robot docking with neural vision and reinforcement," in *Proc. IROS-2003 Workshop on Robot Programming by Demonstration*, Las Vegas, 2003.

[6] C. Gaskett, L. Fletcher, and A. Zelinsky, "Reinforcement learning for visual servoing of a mobile robot," in *Proc. Australian Conf. on Robotics and Automation (ACRA2000)*, 2000.

[7] T. Martínez-Marín and T. Duckett, "Fast reinforcement learning for vision-guided mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA 2005)*, Barcelona, Spain, April 18-22 2005.

[8] R. Sutton, "Reinforcement learning architectures for animats," in *From Animals to Animats, Proc. First Int. Conf. on Simulation of Adaptive Behaviour*, J. Meyer and S. Wilson, Eds. MIT Press, Cambridge MA, 1991.

[9] W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 4, 2002, pp. 3404–3410.

[10] J. Reeds and R. Shepp, "Optimal path for a car that goes both forward and backward," *Pacific J. Math*, vol. 145, no. 2, pp. 367–393, 1990.

[11] M. Spong and M. Vidyasagar, *Robot dynamics and control*. John Wiley and Sons, New York, 1991.

[12] D. Bertsekas, *Neurodynamic Programming*. Athena Scientific, 1996.

[13] A. Moore and C. Atkeson, "Priortized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, pp. 103–130, 1993.

[14] C. Hsu, "A discrete method of optimal control based upon the cell state space concept," *Journal of Optimization Theory and Applications*, vol. 46, no. 4, pp. 547–569, 1985.

[15] P. Zufiria and R. Guttalu, "The adjoining cell mapping and its recursive unraveling, part i: Description of adaptive and recursive algorithms," *Nonlinear Dynamics*, vol. 4, pp. 204–226, 1993.