Cooperative Learning and Planning for Multiple Robots

Sjoerd Van der Zwaan, José A. A. Moreira, Pedro U. Lima
Instituto de Sistemas e Robótica,
Instituto Superior Técnico,
Av. Rovisco Pais,
1049-001 Lisboa,
Portugal
Fax: +351-218418291
E-mail: {sjoerd,moreira,pal}@isr.ist.utl.pt

Abstract

This paper deals with the the subject of learning and planning for real mobile robots, using Sutton's Dyna *algorithm. The* Dyna *algorithm integrates reinforcement learning, planning and reactive execution. In this paper we present an extension of the* Dyna *algorithm which includes symmetric and cooperative learning with multiple robots. We applied the extended version of the algorithm to a population of two real robots. Practical problems associated with the implementation of the algorithm on a real setup are solved. Results obtained from simulations and real experiments are presented and discussed.*

Keywords: Reinforcement Learning, Cooperative Robotics, Visual Tracking, Dyna.

1 Introduction

The control of a robotic system, especially the planning of tasks to be performed by the robot to achieve a goal, is in most cases a very complex problem. Several methods have been developed to address this problem, one of which is reinforcement learning [5]. This method reveals great simplicity, addressing large and complex problems using a simple approach. The learning is done by evaluating the results of executing a given task through success and failure signals and their respective rewards and penalties, allowing the system to plan a solution through a probabilistic decision method by evaluating the performance.

In the last few years, Sutton [1] and his associates have explored reinforcement learning algorithms. One of the algorithms developed is the *Dyna* algorithm, which integrates learning from experiences in the real world and virtual experiences done on an internal world model with reactive execution. The main advantage of this approach is that the hypothetical experiences speed up the learning process. In fact, the algorithm performs an incremental form of planning that is closely related to dynamic programming. Such an approach helps to partially overcome the frequently encountered limitations of reinforcement learning applications to robotics due to the problem large state space. A drawback of this approach is that it assumes that the agent can access the world state at no cost and at every time. This is not always realistic in real robotics applications, e.g. when recognizing the goal state and performing obstacle detection. These are difficult tasks to achieve and certainly not error-free.

In [3], Weiser and Lima present an application of the *Dyna* algorithm to a real robot, in which they study the application of this kind of reinforcement learning algorithms to situations more realistic then the simulations usually described in the literature. The approach adopted consists in navigating a mobile robot through a maze from an initial position to a final position. One conclusion drawn is that with such a setup, initial learning rates are very low, resulting in time consuming first trials. Also, the state representation by a grid of cells is not always suitable, especially when dealing with large state spaces.

In this paper the *Dyna* algorithm is extended to *symmetric* and *cooperative* learning with a population of agents. With this extension, learning rates are speeded-up by exploring cooperation between agents. The extension also explores symmetry in learning data by defining a *foraging and recruiting* behaviour that allows an agent to combine learning data obtained when traveling from an initial state to a goal state and vice versa. We present an application of the extended algorithm, using a real set-up with two robots.

The Section 2 outlines the *Dyna* algorithm and then extends the algorithm to symmetric and cooperative learning. In section 3, an experimental setup to apply the extended *Dyna* algorithm to a set of real robots is described. Section 4 presents some results obtained from a series of experiments performed with the extended algorithm. Finally, in Section 5, conclusions are drawn and future work is discussed.

2 The Extended Algorithm

2.1 The *Dyna* algorithm

The *Dyna* algorithm [1],[2] is based on the old idea that planning is like trial and error learning from hypothetical experiences. Based on this concept, an agent interacts with the world, from which it receives a success or failure signal associated to a given state-action pair. Together with the success signal, the agent receives a reward. The agent also maintains an internal model of the world that is updated with the information gathered by interacting with the real world. As an additional feature, the agent performs hypothetical experiments using the current world model, intermixed with the interaction with the real world.

Together with the world model, there also exists a policy map that associates a set of possible actions to each state. Each state in the world has an evaluation value associated to it, which is updated from the reinforcement signals received from the world after each action. This is done according to the simplest version of the temporal difference learning method [5]. The algorithm uses this evaluation to update the policy so that the agent is able to plan the correct sequence of actions to achieve the goal. The policy table has an entry w_{xa} for every pair of state x and action a, which is updated (using the evaluation function) so as to strengthen or weaken the tendency to perform action a in state x. Actions are chosen randomly according to a Boltzmann probability distribution, so as to guarantee stochastic convergence of the algorithm:

$$p\{a \mid x\} = \frac{e^{w_{xa}}}{\sum_j^{actions} e^{w_{xj}}} \tag{1}$$

A step is defined as the transition from one cell to a neighboring cell. The algorithm combines real steps, that are performed by the agent in the real world, with hypothetical steps that are executed by the agent over its world model. Both types of steps update the evaluation function and the policy map of the agent. A trial is defined as a sequence of real steps which achieves the goal state starting at some initial state. For a detailed description of the *Dyna* algorithm we refer to [2].

2.2 The symmetric problem

Based on the above version of the *Dyna* algorithm, the agent needs to return to its initial position and start another trial each time it reaches a goal state. This calls for the need of an external agent that places the learning agent in its initial position or alternatively the need to retrace the steps that were necessary to reach the goal. Observing the dual between the path traveled by the agent when stepping from the initial position to the goal position and vice versa, we extended the *Dyna* algorithm such that the agent is able to continue learning when traveling from the goal state to its initial position. This solves the problem of returning the agent off-line to the initial position.

For the *Dyna* algorithm to work in this *symmetric* way, it is necessary to maintain two evaluation functions (one to travel from the initial position to the goal and another on the way back), so as to preserve the mechanism of backpropagation of the evaluation function values in each state. Symmetric information can be found in the policy map, which indicates stepping directions in each state.

The main idea behind the extension is that after entering the goal state, it is possible to transform the current policy such that it directs the agent back to the initial position instead of the goal position. Considering the case in which the algorithm is applied to solve a maze as in [1], where the state is described by the position of the agent in the maze given by a coordinate pair (i, j) with $i \equiv rows = 0, 1, ..., n-1$ and $j \equiv columns = 0, 1, ..., m-1$ and where the set of available actions in each state is given by: $action \in (left, right, up, down)$, we propose the following scheme to implement such a transformation:

For $i = 0 : n-1$
    For $j = 0 : m-1$
        If state (i, j) is not occupied by an obstacle
        Then:
            if cell $(i, j-1)$ has no obstacle
                $policy(i, j)_{left} = w(i, j-1)_{right}$
            else
                $policy(i, j)_{left} = w(i, j)_{left}$
            if cell $(i, j+1)$ has no obstacle
                $policy(i, j)_{right} = w(i, j+1)_{left}$
            else
                $policy(i, j)_{right} = w(i, j)_{right}$
            if cell $(i+1, j)$ has no obstacle
                $policy(i, j)_{up} = w(i+1, j)_{down}$
            else
                $policy(i, j)_{up} = w(i, j)_{up}$
            if cell $(i-1, j)$ has no obstacle
                $policy(i, j)_{down} = w(i-1, j)_{up}$
            else
                $policy(i, j)_{down} = w(i, j)_{down}$
For $i = 0 : n-1$
    For $j = 0 : m-1$
        $w(i, j)_{left} = policy(i, j)_{left}$

$$w(i,j)_{right} = policy(i,j)_{right}$$
$$w(i,j)_{up} = policy(i,j)_{up}$$
$$w(i,j)_{down} = policy(i,j)_{down}$$

Where w_{xa} is the entry of the policy table for the state-action pair given by state $x = (i,j)$ and action a and $policy(i,j)_a$ is an auxiliary variable used to calculate the transformed values for w_{xa}. The value of w_{xa} is used to compute the probability of an agent in the state (i,j) to move in the direction given by a, according to (1). Note that in a given state (i,j), the value of the policy towards an obstacle, known in the world model, is not changed by the transformation.

For the agent to be able to add new information to the policy table when returning to the initial position, it is necessary to attribute a reward to the initial position and reset the reward in the goal position. Using the second evaluation map when returning to the initial position, it is now possible to apply the classical *Dyna* algorithm with the transformed policy and iterate until the agent reaches its initial position. At this point the policy map is transformed again, a reward is attributed to the goal position and using the first evaluation function, the algorithm now iterates again until entering the goal state.

The only problem that arises concerns the use of the two separate evaluations functions. When the agent steps back to its initial position using the transformed policy and the second evaluation function, a situation is created in which increased policy values appear in states for which no increased evaluation exists. But taking into account the structure of the *Dyna* algorithm, the hypothetical and real steps that will be performed will make the second evaluation function converge to a symmetric version of the first evaluation map.

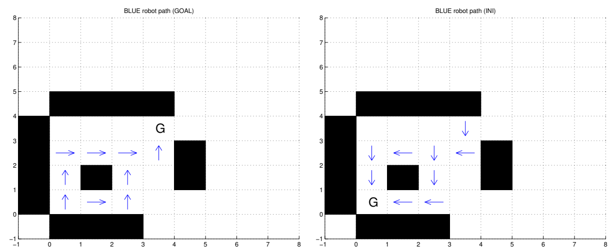

Figure 1: Example of the policy map before (left) and after (right) applying the transformation; the arrows represent the direction containing the largest policy values; no arrow means that all directions have a zero or negative policy value; the goal state is at (3,3) and the initial state at (0,0).

2.3 Cooperative learning and planning

Another interesting extension for the *Dyna* algorithm is to have a population of small robots running in parallel through the maze, with the objective of speeding up the learning algorithm and testing cooperative strategies. Extending the *Dyna* algorithm for a cooperative set of agents means information sharing between the agents. In the case of communicating agents, it is also necessary to consider the communication channel between real agents as not completely reliable and noisy. Taking into account these considerations, we propose the following cooperation strategy:

- Each agent maintains its own internal world model , evaluation function and policy map.
- Upon discovering a new obstacle or goal, an agent transmits the position of that obstacle or goal to the rest of the population.
- Upon taking a real transition between states, each agent transmits the resultant policy value obtained for that specific state-action pair to the rest of the population. The receiving agents update the corresponding entry of their policy map using the following expression:

  $$w(i,j)_{action} = \frac{w(i,j)_{action} + value}{2}$$

  Where *value* is the resulting policy value transmitted by the agent that performed the transition.

The main idea behind communicating the policy values is that sharing the information of the new obtained policy value for a given state-action pair with the other agents will reduce the search space of the other agents, thus accelerating the learning process. Note that although initially there will be a discrepancy between the evaluation and policy maps of the receiving agents, the hypothetical experiments that are performed by each agent will reduce the discrepancy.

Another advantage of this cooperation strategy is its robustness. Since each agent has its individual world model, it can always perform individual learning and planning. Then, if for some reason a communication failure arises during an interval of time, each agent still is capable of fulfilling its objectives on a stand-alone basis.

A disadvantage is that each agent needs a significant computational capability to execute the *Dyna* algorithm for which it also needs some memory to be able to represent the world model with corresponding policy maps and evaluation functions. An alternative is to assign some processing time and memory of a central processing unit to each individual agent which runs the individual learning and planning modules in parallel. This way, the agents only receive the actions and transmit the results of those actions to the central unit. This approach will be further explored in the following section, which deals with the implementation of the extended algorithm on a real system.

2.4 Deadlock avoidance

When running the extended *Dyna* algorithm for a population of cooperative agents that share the same goal, all individual policies will converge to the same policy due to the backpropagation of the common received rewards. This creates situations in which the agents will plan transitions to common states, leading to deadlocks. A deadlock occurs whenever two or more agents prepare a step in the real world to the same state or whenever two or more agents are face to face and plan to swap positions. To solve the deadlock situations, we propose an alternative state transition rule, triggered whenever a deadlock occurs. According to this rule , instead of consulting its policy, an agent selects a random action with equal probability from the set of available actions. If the action is executable, the agent makes the transition without changing the policy of the corresponding state-action pair and without updating the evaluation values of the corresponding state.

3 Experimental Setup

3.1 Overall system

This section describes the experimental setup used for testing the extended version of the *Dyna* algorithm on real robots. The overall system, illustrated in Figure 2, consists of two vision-based teleoperated cellular robots [6] controlled via a camera located at an elevated position such that the camera-image covers the whole workspace of the robots.

The robots have no on-board sensors and are controlled by a central processing unit. For each individual robot, the central computer runs three modules: the learning and planning algorithm, a visual tracking module and a control module. The visual tracking module is responsible for robot pose estimation from the visual information provided by the camera. The control module controls the individual robot position and heading direction via a radio link. Each module will be described in more detail in the following subsections.

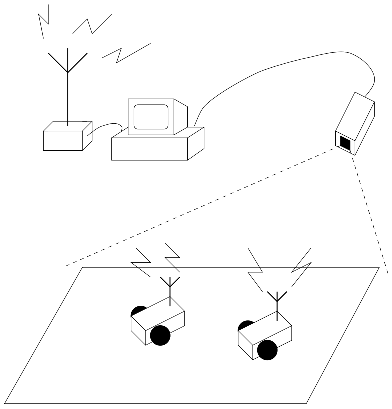

Figure 2: Schematics of the overall system used.

3.2 Learning and planning for real robots

To apply the extended version of the *Dyna* algorithm to real robots, the world is defined as a grid of square cells (states), projected onto the ground-plane. Obstacles are simulated by black cells. In Figure 3, this real setup is illustrated.

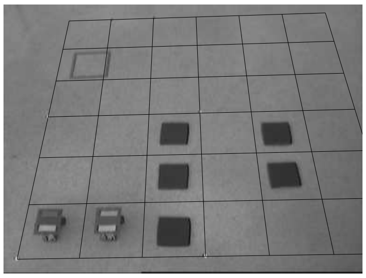

Figure 3: Example of a real world setup; the maze is super-imposed on the camera image.

Initially each individual robot has an internal representation of the world (stored in the central computer), which is empty. For each robot step in the real world, the learning and planning algorithm provides a reference input obtained from the corresponding robot policy. To execute the step, each robot first checks the visual information provided by the camera to determine if the reference cell is empty. Although the camera provides the central computer with visual information concerning the whole workspace of the robots, the individual robot does not access this information. In this setup, on-board vision sensors are simulated, by allowing each robot only to scan for obstacles or other robots in the direction of movement.

Upon successful execution of a step in the real world, the corresponding world model is updated.

3.3 Visual sensing and tracking

In order to control the trajectory of each individual robot, it is necessary to relate the robot position on the ground-plane (in metric coordinates) to its position in the image-plane (given in pixel coordinates). Each image point $\tilde{m}$ will correspond uniquely to a certain point on the ground-plane $\tilde{M}$, according to a plane-to-plane projective transformation [7]:

$$\tilde{m} = \tilde{P}_p.\tilde{M}, \tag{2}$$

$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3}$$

The transformation matrix P depends on the camera intrinsic parameters and the camera position relative to the world frame. Once the transformation matrix is estimated, it can be used to convert the coordinates from robots and obstacles in the image plane to the ground-plane and vice versa.

A tracking system is developed which estimates each individual robot position and heading direction over time from the sequence of camera images. The video camera uses a RGB representation, allowing color detection for robot segmentation in the image-plane. The two robots used in this setup are then distinguished by their color (a blue robot and a red robot). To make the tracking system more robust, the brightness information is removed from the original RGB-image by a normalization process [4]. This normalization transforms the RGB-image into chromatic color space, resulting in an intensity image for each color of the triple [r,g,b]. For instance to detect the blue robot, a threshold is applied to the image normalized in blue, which transforms this image into a binary image. The robot position in the image-plane is obtained by calculating the center-off-mass in the binary image. This is illustrated in Figure 4.

Figure 4: Robot position estimation in the image-plane; at the left one finds the original RGB-image; the middle image shows the normalized image; thresholding the normalized image results in the binary image shown at the right.

In a similar way, the robot heading direction is estimated by the segmentation of two colored bars situated on top of the robot and of different colors as the robot itself. The direction of the vector passing through the centers-off-mass of these bars gives an estimate of the instantaneous robot heading direction. Assigning different colors to the bars, the robot's front- and back sides can be identified.

The tracking systems runs at a frequency of about 5 Hz and performs robust estimation under various lightning conditions. Image processing is done locally in a small neighborhood of the actual robot position.

3.4 Robot control system

The control system implemented on the central computer runs a control algorithm for each individual robot, controlling its position and orientation towards a final position provided by the learning and planning algorithm.

The mobile platforms used have a differential-drive structure, where two DC-motors directly drive the left and right wheels independently. A kinematic model for the differential-drive structure can be obtained by considering a coordinate frame attached to the midpoint, C, in the wheels axis. The linear and angular velocities of C, V_c and ω_c, can be expressed as a function of the angular velocities of the left and right wheels ω_L and ω_R, the half-distance D between the wheels , and the radius R of the wheels, according to:

$$\begin{bmatrix} V_c \\ \omega_c \end{bmatrix} = \begin{bmatrix} \frac{R}{2} & \frac{R}{2} \\ -\frac{R}{D} & \frac{R}{D} \end{bmatrix} \cdot \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} \tag{4}$$

To obtain a dynamic model, we consider the DC-motor. The transfer function between the motor shaft velocity and the input voltage is approximated by:

$$\omega(s) = \frac{K_\omega}{\tau.s+1}.U(s) \tag{5}$$

where τ is the time constant of the dominant mechanical pole and K_ω is the steady-state gain. Using the kinematic model given by eq. (4), it is possible to obtain the transfer functions between the left and right motor input voltages and the angular and linear velocities of the vehicle reference point C:

$$\omega_C(s) = \frac{R}{D}.\frac{K_\omega}{\tau.s+1}.(U_R(s) - U_L(s)), \tag{6}$$

$$V_c(s) = \frac{R}{2}.\frac{K_\omega}{\tau.s+1}.(U_R(s) + U_L(s)) \tag{7}$$

The control strategy consists in dynamically orienting the robot towards the final position, specified by the center position of the goal cell and provided by the learning and planning module. The controller receives the actual position and orientation sensed by the visual system and generates appropriate motor commands. Along the path, the robot will move at constant cruise speed. The robot position is controlled by an on/off controller, generating a constant common-mode voltage (resulting in a constant linear velocity of the robot) whenever the robot is outside a predefined radius encircling the final position. A PID-controller is used to control the robot heading direction, generating differential voltages as a function of the error in orientation. This differential signal is super-imposed on the common mode signal and sent to each robot by a radio-link. The radio-link operates at a rate of 1200 bps via serial-port communication with the central computer. At this rate, the central unit reaches a control frequency up to 21.8 Hz with a single robot and 1.36 Hz with 16 robots.

4 Results

In this section we compare the performance of the *symmetric* single agent algorithm with the *cooperative symmetric* algorithm. Results are obtained from experiments with the real world setup as illustrated in Figure 3.

The real-time experiments were performed using the symmetric algorithm for a single robot (blue robot) and the cooperative symmetric algorithm for a set of two robots (red and blue robot). In the second case, the blue robot was placed in the initial position and the red robot was placed next to the blue robot. Nevertheless it could be placed anywhere in the world. The parameter setting used for the *Dyna* algorithm is: $\alpha = 4, \quad \beta = 0.1, \quad \gamma = 0.9$. One hundred hypothetical steps were performed for each real step. Evaluation values were initialized at zero.

The performance measure used for comparison is the number of steps per trial. In the original *Dyna* algorithm a trial is defined as a complete path from the initial position to the goal position. Since with the *symmetric* algorithm the agent will also learn when returning from the goal position to the initial position, it is necessary to restate the definition of a trial so as to be able to compare performances. Defining a trial as a complete path between the initial position and goal position or vice versa, we used the average value between the number of steps obtained from stepping from the initial position to the goal position and the number of steps obtained from stepping back from the goal position to the initial position.

Figure 5 displays the average number of steps per trial obtained from three experiments with both the symmetric and cooperative symmetric algorithm, where each experiment runs five trials. Also the average number of deadlocks that occurred with the cooperative symmetric algorithm are displayed.

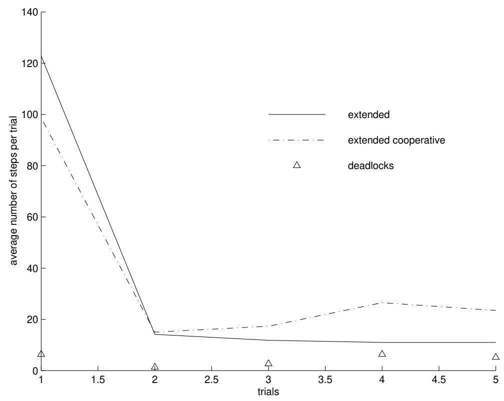

Figure 5: Results obtained from real experiences. The values indicated are average values obtained from 5 experiences.

Analyzing the obtained results it is possible to verify that the experiment with two robots will initially converge faster to the solution than the experiment with a single robot. This is also illustrated in Figure 6, where the internal world model and policy of the blue robot upon entering the goal state for the first time is illustrated for both the symmetric- and cooperative symmetric algorithm. With the cooperative algorithm, the blue robot policy is much more complete due to information exchange with the red robot, which already has entered the goal state.

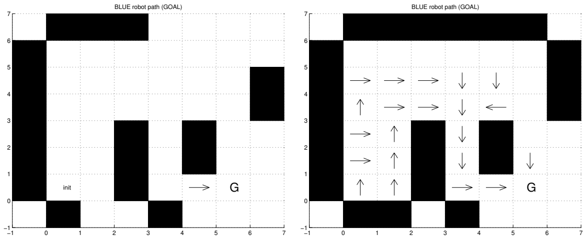

Figure 6: Internal world model and obtained policy of the blue robot upon entering the goal state for the first time; left: symmetric algorithm; right: cooperative symmetric algorithm.

An important observation is that due to deadlock situations, that occur in the cooperative setting, the number of steps per trial will oscillate, while in the case of a single robot, the number of steps converged to a minimum after three trials. It is important to realize that the oscillation of the number of steps per trial for the cooperative setting does not imply that the path planned by the robots changes. The increased value of steps is due to the extra steps needed to solve the deadlock.

5 Conclusions and Future Work

The results obtained with the symmetric *Dyna* algorithm show that the use of a cooperative set of agents allows the algorithm to reach the goal faster, especially in the first iteration where no knowledge of the world and of the goal position is available. The possibility of distributing the agents over the world allows the algorithm to reach the goal even faster, since each agent will explore a different region of the search space and transmit that information to all other agents.

The development of a symmetric algorithm allows the agents to return to their initial position while continuing to learn.

A deeper experimental study using larger worlds is necessary to demonstrate more clearly the performance of the cooperative extended algorithm. A theoretical study must be made to show the stochastic convergence of this extended *Dyna* algorithm.

Future planned improvements include the usage of fully autonomous robots, without the need to use an overviewing camera and external processing.

Acknowledgments

We would like to thank José Santos-Victor for the use of the installations and equipment of the Computer and Robot Vision Lab - Vislab - at the Institute of Systems and Robotics.

References

[1] Richard S. Sutton, "First Results with Dyna, an Integrated Architecture for Learning, Planning and Reacting," *Neural Networks for Control*, The MIT Press, 1990.

[2] Richard S. Sutton, "Dyna, an Integrated Architecture for learning,Planning, and Reacting," *Working Notes for the 1991 AAAI Spring Symposium*, pp. 151-155, 1991.

[3] Alex Weiser and Pedro Lima, "An Integrated Learning, Planning and Reacting Algorithm Applied to a Real Mobile Robot," *in Proceedings of Controlo'96*, Portugal, 1996.

[4] Jie Yang and Alex Waibel, "A Real-Time Face Tracker," *Proceedings of the WACV 96*, Florida, 1996.

[5] Tom M. Mitchell, "Machine Learning," , Mc.Graw Hill 1987.

[6] José Santos Victor, " Vision-based remote control of cellular robots," , *Robotics and Autonomous Systems*, 23, pp 221-234, 1998.

[7] B. Horn, "Robot Vision," , MIT Press 1986.