WORLD EXPLORATION AND PATH LEARNING WITH A KHEPERA MOBILE ROBOT

Rui Araújo, and A. T. de Almeida

Institute of Systems and Robotics (ISR); and
Electrical Engineering Department; University of Coimbra;
Pólo II; Pinhal de Marrocos; 3030 Coimbra - Portugal;
Phone: +351 (39) 7006276/00; Fax: +351 (39) 7006247; email: rui@isr.uc.pt

ABSTRACT

This article presents a robust approach to learn motion trajectories of a mobile robot navigating through an environment. The location and shape of obstacles present in the robot world is initially unknown. The robot performs obstacle detection using its infrared distance sensors. This enables the system to explore, and construct a model of the environment that is based on a multiresolution partitioning of the world. Trajectory learning is performed using the information present on the learned world model. The results of experiments concerning the application of the learning approach to a mobile robot are presented.

Keywords: Exploration, mobile robot, learning.

1. Introduction

A basic issue a mobile robot has to solve is the path finding problem where it must plan to execute collision-free motions. The successful navigation of a mobile robot on an unknown environment is greatly dependent on the ability to construct a model of its environment. In this article we face the problem of controlling an autonomous mobile robot, so that it becomes able to navigate through an obstacle-free path.

Many control methods are based on mathematically modelling the system/plant to be controlled. In the case of mobile robotics for example, it is usual to assume world knowledge, generally appearing in the form of a global map of the world over which path planning algorithms operate. However it is difficult to provide the robot with a global map model of its world. The difficulties may arise from various reasons. The map building operation is itself a difficult separate problem. Also, if the robot control system requires the introduction of the global map, this may become a tedious and time consuming programming task. On the other hand, reactive systems (e.g. [1]) don't require the existence of a global map. Those systems are organised as a layered set of task-achieving modules. Each module implements one specific control strategy or behaviour like “avoid hitting anything” or “keep following the wall”. Each behaviour implements a close mapping between sensory information and actuation to the system. However, basic reactive systems suffer from two shortcomings. First, they are difficult to program. Second and most important, pure reactive controllers may generate inefficient trajectories since they choose the next action as a function of the current sensor readings, and the robot's perception range is limited. To address this second
problem, some control architectures combine a priori knowledge and planning, with reaction (e.g. [2]).

In this paper we demonstrate the effectiveness of the parti-game algorithm [3] to the specific case of learning a mobile robot path from its initial position, to a known goal region in the world. Also the algorithm does not have any initial internal representation, map, or model, of the world. We demonstrate that the mobile robot can learn to navigate to the goal having only the predefined abilities of doing straight-line motion, and obstacle detection (not avoidance) using its own distance sensors. There is no explicit split between a learning phase and an execution phase. The robot is always planning to move towards the goal with the information available on its current map. At the same time as the robot faces new situations and receives sensory data from its sensors, it improves the internal model of the world. This in turn will enable the improvement of the motion planning solutions of the robot. In general terms a learning system may be viewed as bringing a beneficial concept of self programming, in which control of a complex system in principle does not need extensive analysis, modelling, programming, assistance or teaching by human experts.

The organisation of the paper is as follows. Section 2 presents the learning controller architecture. In section 3 we present the experimental environment around which we performed some experiments concerning the application of the exploration algorithm. Section 4 presents results of simulations using the controller for learning to navigate the robot to a goal region on an unknown world. Finally in section 5 we make some concluding remarks.

2. Learning Controller Architecture

The problem we wish to solve in this work may be defined as follows: The mobile robot is initially on some position on an unknown environment, and then it must learn to navigate, through an obstacle-free path, to a known goal region in the world. The algorithm does not have any initial internal representation, map, or model, of the world. In applying the algorithm, we assume that the initial abilities of the mobile robot are only two. First, it is able to perform obstacle detection operations, i.e. to detect obstacles that may obstruct its normal path. Second, the mobile robot is able to perform straight line movements between its current position and some other specified position in the world. Two comments to this second ability are in order. First, this ability requires the assumption that the robot knows its own current position and orientation, and second, the movements may fail because of the presence of an obstacle that is detected. The algorithm assumes that a local greedy controller is available, which we can be asked to move the system greedily towards a desired state. However, there is no guarantee that a request to the greedy controller will succeed. For example, in this work the greedy controller is the "straight-line mover", which can possibly become stuck due to an obstacle that is detected to be obstructing the robot path.

The mobile robot controller architecture used in this work, is based on the application of the parti-game learning controller algorithm [3] to the specific case of learning a mobile robot path. The algorithm can be applied to learning control problems in which: (1) We have continuous and multidimensional state and action spaces; (2) "Greedy" and hill-climbing techniques can become stuck, never reaching the goal; (3) random exploration can be intractably time-consuming; (4) We have unknown, and possibly discontinuous, system dynamics and control laws. Also, the algorithm assumes that a local greedy controller is available, which we can ask to move greedily towards a desired state. However, there is no guarantee that a request to the greedy controller will succeed. On our case for example, it is possible for the greedy controller (the "straight-line mover") to become stuck because of an obstacle found by the robot.

The parti-game algorithm is based on partitioning the state-space. It begins with a large partition. Then it increases the resolution by subdividing the state-space (see figure 1) where the learner predicts that a higher resolution is needed. As usual a partitioning of the state-space is a finite set of disjoint regions, the union of which covers the entire state-space. Those regions will be called
cells, and will be labelled with integers $1, 2, \ldots, N$. In this paper we will assume that the cells are all axis-aligned hyperrectangles (rectangles in our 2D case). Although this assumption is not strictly necessary, it simplifies the computational implementation of the algorithm. A real-valued state, $s$, is a vector of real numbers in a multidimensional space (2D space in our case). Real-valued states and cells are distinct entities. For example the right partitioning on figure 1 is composed of eleven cells, which are labelled with numbers $1 \ldots 11$. Each real-valued state is in one cell and each cell contains a continuous set of real-valued states. Let us define $\text{NEIGHS}(i)$ as the set of neighbours, or cells which are adjacent to $i$. In figure 1 $\text{NEIGHS}(1) = \{2, 3, 4\}$. When we are at a cell $i$, applying an action consists on actuating the local greedy controller “aiming at cell $j$”. A cell $i$ has an associated set of possible actions that is defined as $\text{NEIGHS}(i)$. Each action can thus be labelled by a neighbouring cell.

The algorithm uses an environmental model, which can be any model (for example, dynamic or geometric) that we can use to tell us for any real-valued state, control action, and time interval, what will be the subsequent real-valued state. In our case the “model” is implemented by the mobile robot (which can be real or simulated), and takes the current position, and position command, to generate the next robot position.

Let us define the $\text{NEXT-PARTITION}(s, j)$ function that tells us in which cell we end up, if we start at a given real-valued state, $s$, and using the local greedy controller, keep moving toward the centre of a given cell, $j$, until either we exit the initial cell or get stuck. Let $i$ be the cell containing the real-valued state $s$. If we apply the local greedy controller “aim at cell $j$” until either cell $i$ is exited or we become permanently stuck in $i$, then

$$\text{NEXT-PARTITION}(s, j) = \begin{cases} i & \text{if we became stuck} \\ \text{the cell containing the exit state} & \text{otherwise} \end{cases}$$

In our work, the test for sticking performs an obstacle detection with the distance sensors of the mobile robot (see section 3). In other systems, sticking could be tested by seeing if the system has not exited the cell after a predefined time interval.

Let $\text{CENTER}(i)$ be the real valued state at the centre of cell $i$. Define $\text{NEXT}(i, k)$ as the cell where we arrive if, starting at $\text{CENTER}(i)$, we apply the local greedy controller “aiming at cell $k$”.

$$\text{NEXT}(i, k) = \text{NEXT-PARTITION}($$ \text{CENTER}(i), k$$)$$

In general $\text{NEXT}(i, k) \neq k$ because the local greedy controller is not guaranteed to succeed. Since there is no guarantee that a request to the local greedy controller will succeed, each action has a set of possible outcomes. The particular outcome of an action depends on the real-valued state $s$, from which we start “aiming”. The outcomes set, of an action $j$ in cell $i$, is defined as the set of possible next cells:

$$\text{OUTCOMES}(i, j) = \{ k \mid \text{exists a real-valued state } s \text{ in cell } i \text{ for which } \text{NEXT-PARTITION}(s, j) = k \}$$

We can now, using a worst case assumption, define the shortest path from cell $i$ to the goal, $J_{WC}(i)$, as the minimum number of cell transitions to reach the goal assuming that, when we are in a certain cell $i$ and our intended next cell is $j$, an adversary is allowed to place us in the worst
**Algorithm 1**

REPEAT FOREVER.

1. FOR each cell \( i \) and each neighbour \( j \in \text{NEIGHS}(i) \), compute the OUTCOMES\((i, j)\) set in the following way:

   1.1 IF there exists some \( k' \) for which \( (i, j, k') \in D \) THEN:

       \[
       \text{OUTCOMES}(i, j) = \{ k \mid (i, j, k) \in D \}
       \]

   1.2 ELSE, use the optimistic assumption in the absence of experience:

       \[
       \text{OUTCOMES}(i, j) = \{ j \}
       \]

2. Compute \( J_{WC}(i') \) for each cell using minimax.

3. Let \( i := \) the cell containing the current real-valued state \( s \).

4. IF \( i = \text{GOAL} \) THEN exit, signalling SUCCESS.

5. IF \( J_{WC}(i) = \infty \) THEN exit, signalling FAILURE.

6. ELSE

   6.1 Let \( j := \arg\min_{j' \in \text{NEIGHS}(i)} \max_{k \in \text{OUTCOMES}(i, j')} J_{WC}(k) \)

   6.2 WHILE (not stuck and \( s \) is still in cell \( i \))

       6.2.1 Actuate local greedy controller aiming at \( j \).

       6.2.2 \( s := \) new real-valued state.

   6.3 Let \( i_{\text{new}} := \) the identifier of the cell containing \( s \).

   6.4 \( D := D \cup \{(i, j, i_{\text{new}})\} \)

LOOP

Figure 2: Algorithm 1.

position within cell \( i \) prior to the local controller being activated. The \( J_{WC}(i) \) shortest-path is defined as:

\[
J_{WC}(i) = \begin{cases} 
0, & \text{if } i = \text{GOAL} \\
1 + \min_{k \in \text{NEIGHS}(i)} \max_{j \in \text{OUTCOMES}(i, k)} J_{WC}(j), & \text{Otherwise}
\end{cases}
\]

(1)

The \( J_{WC}(i) \) values can be obtained by the minimax algorithm [4] or by Dynamic Programming. The value of \( J_{WC}(i) \) can be \(+\infty\) if, when we are at cell \( i \), our adversary can permanently prevent us from reaching the goal. By definition such a cell is called a losing cell. With this method, the next cell to aim is the neighbour, \( i \), with the lowest \( J_{WC}(i) \). Using this approach we are sure that, if \( J_{WC}(i) = n \), then we will get \( n \) or fewer transitions to get to the goal starting from cell \( i \). However, the method is too much pessimistic because, regions of a cell that will never be actually visited, are available for the adversary to place us. But those may be precisely the regions that lead to an eventual failure of the process. So although this method guarantees success if it finds a solution, it may often fail on solvable problems.

Next we will describe Algorithm 1, that reduces the severity of this problem by considering only all empirically observed outcomes, instead of all possible outcomes for a given cell. Another argument contributing to this solution, is that as a learning algorithm, it is more important to learn the outcomes set, only from real experience on the behaviour of the system. Besides that, it could be difficult or impossible, to compute all possible outcomes of an action. Whenever an \( \text{OUTCOMES}(i, j) \) set is altered due to a new experience obtained, equation (1) is again solved in order to find the path to the goal. Before an action is experienced, we can not leave the \( \text{OUTCOMES}(i, j) \) set empty. In these situations we use, the default optimistic assumption that
ALGORITHM 2
WHILE (s is not in the goal cell )

1. Run Algorithm 1 on s and P. Algorithm 1 returns the updated database D, the new real-valued state s, and the success/failure signal.

2. IF FAILURE was signalled THEN
   2.1 Let $Q :=$ All losing cells in P ($J_{WC} = \infty$).
   2.2 Let $Q' :=$ The members of $Q$ who have any non-losing neighbours.
   2.3 Let $Q'' := Q'$ and all non-losing members of $Q'$.
   2.4 Split each cell of $Q''$ in half along its longest axis producing a new set $R$, of twice the cardinality.
   2.5 $D := D + R - Q''$
   2.6 Recompute all new neighbour relations, and delete from the database D, those triplets that contain a member of $Q''$ as a start point, an aim-for, or an actual outcome.

LOOP

Figure 3: Algorithm 2.

we can reach the neighbour that is aimed. Algorithm 1 (see fig. 2) keeps applying the local greedy controller, aiming at the next cell, on the “minimax shortest path” to the goal, until either we are caught on a losing cell ($J_{WC} = \infty$), or reach the goal cell. Whenever a new outcome is experienced, the system updates the corresponding OUTCOMES(i,j), and equation (1) is solved, to obtain the, possibly new, “minimax shortest path”. Step 6.1 computes the next neighbouring cell on the “minimax shortest path” to the goal. Algorithm 1 has three inputs: (1) The current (on entry) real-valued state s; (2) A partitioning of the state-space, P; (3) A database, the set $D$, of all previously different cell transitions observed in the lifetime of the partitioning P. This is a set of triplets of the following form: (start-cell, aimed-cell, actually-attained-cell). At the end Algorithm 1 returns three outputs: (1) The updated database of observed outcomes, D, (2) the final, real-valued system-state s, and (3) a boolean variable indicating SUCCESS or FAILURE.

We see that Algorithm 1 gives up when it discovers it is in a losing cell. One of the hypothesis of the algorithm is that all paths through the state space are continuous. Assuming that a path to the goal actually exists through the state-space, i.e. the problem is solvable, then there must be an escaping-hole allowing the transition to a non-losing cell and eventually opening the way to reach the goal. This hole has been missed by Algorithm 1 by the lack of resolution of the partition. A hole for making the required transition to a non-losing cell, can certainly be found on the cells at the borders between losing and non-losing cells. Taking this comments into account, the top level Algorithm 2 (see fig. 3), divides in two the cells in the borders, in order to increase the partition resolution, and to allow the search for the mentioned escaping-hole. This partition subdivision takes place between, each successive calls to Algorithm 1 that keep taking place while the system does not reach the goal region. Algorithm 2 has three inputs: (1) The current (on entry) real-valued state s; (2) A partitioning of the state-space, P; (3) A database, the set $D$, of all previously different cell transitions, observed in the lifetime of the partitioning P. This is a set of triplets of the form: (starting-cell, aimed-cell, actually-attained-cell). At the end, Algorithm 2 returns two outputs: (1) The new partitioning of the state-space, and (2) a new database of outcomes $D$.

3. Experimental Environment

In this section we give details of the experimental environment that we used to prove, the effectiveness of the algorithm presented in section 2 to solve the robot path finding problem that was formulated on the same section 2. This work is based on the Khepera miniature mobile robot [5], [6]. The circular shaped Khepera mobile robot (see figure 4) has two wheels, each controlled by a DC motor that has an incremental encoder and can rotate in both directions. Each motor can take a speed ranging from -10 to +10. The unit is the (encoder pulse)/10ms that corresponds to
8 millimetres per second. Additionally, for physical supporting purposes the robot has two small balls placed under its platform. The mobile robot includes eight infrared proximity sensors placed around its body, two pointing to the front, four pointing to the left and right sides, and two pointing to the back side of the robot – see figure 4. Each proximity sensor is based on the emission and reception of infrared light. Each receptor is able to measure both the ambient infrared light (with the emitter turned off) and the reflected infrared light that was emitted by the robot itself. The sensor measures do not have a linear characteristics and depend on external factors such as objects surface properties, and illumination conditions. Each sensor reading returns a value between 0 and 1023. A value of zero means that no object is perceived, while a value of 1023 means that an object is very close and almost touching the sensor. Intermediate sensor values may give an approximate idea of the distance between the sensor and the external object.

The results reported in this paper were achieved using the “Khepera Simulator Version 2.0” [7]. This simulator allows a Khepera robot to evolve on a square world of 1000 mm of side width, where an environment for the robot may be created by disposing bricks, corks and lamps. To calculate its output distance value, a simulated robot sensor explores a set of 15 points in a triangle in front of it. An output value is computed as a function of the presence, or absence, of obstacles at these points. A random noise corresponding to ±10% of its amplitude is added to the output light value. The light value output of a sensor is computed as a function of both the distance and the angle between the sensor and the light source. A ±5% random noise is added to the resulting value. In our simulations we do not use corks and lamps, but only bricks. Also we do not use the light value reading from the sensors but only the distance value.

For the sticking condition test primitive, that is required in the algorithm described in section 2, we use \( d(0), \ldots, d(5) \), which are the distance values of robot sensors 0 through 5 respectively (front – see figure 4). After performing tests with the mobile robot we concluded that it is appropriate to consider that the robot was stuck if at least one of the distances \( d(0), \ldots, d(5) \) increases above 700, 400, 900, 900, 400, 700 respectively. With respect to the motors, the simulated robot simply moves accordingly to the speed set by the algorithm of the user program. A random noise of ±10% is added to the amplitude of the motor speed, and a ±5% random noise is added to the change of direction (angle) resulting from the difference of speeds of the motors.

4. Experimental Results

In this section we present results of simulation experiments, regarding the application of the learning algorithm described in section 2. In the presented experiment, we want to solve the problem of finding a mobile robot path to a predefined goal region. While learning a path, the robot uses its sensors to detect obstacles on the way.

In the experiment there are some walls on the mobile robot’s world that make it difficult to travel from the start position to the goal region (see figures 5, 6, 7, 8, and 10). In figure 5 we can
observe the trajectory of the mobile robot on the first trial to reach the goal. As can be seen the robot does lots of exploration in this first trial. This is because the robot does not have any initial knowledge about the world. In spite of this, the robot is already able to reach the goal in the first trial. As the robot performs more trials it accumulates knowledge about the map of the world. In trials 2 and 3 (see figures 6 and 7) there are wide regions where the robot has no need to explore anymore. However we see that the robot still performs a thorough exploration of the first wall it faces when going to the goal. This is because, in an attempt to find a narrower opening to the goal, the robot is increasing the resolution on that area. This would enable the establishment of a shorter path to the goal that would avoid going round the first wall. This refinement near the first wall can also be observed by comparing the partition at the end of the first trial (figure 9) and the final partition (figure 10). At the beginning of the fourth trial of the experiment, the robot does no further knowledge acquisition. It is already able to promptly navigate, through an obstacle-free path to the goal – see figure 8. The final path even though not being optimal (e.g. in terms of shortest geometric travelling path), can subjectively be said to be “weakly optimal” and certainly “not strongly suboptimal”. The corresponding final partition can be observed in figure 10. We can see that this partitioning is a representation of the world suitable to the learning algorithm. The robot increases the resolution of the partition on areas where it faces greater difficulties to navigate. We also see that the robot is able to avoid exploring many unnecessary areas.
5. Conclusion

In this article we have demonstrated the validity of a learning approach for navigating a mobile robot, by finding a path to a goal region of an unknown environment. Initially, the robot has no map of the world, and has only the abilities of sensor-based obstacle detection and straight-line movement. Simulation results demonstrating the application of the learning approach to a mobile robot were presented.

References


ACTAS 5
JORNADAS HISPANO-LUSAS DE INGENIERÍA ELÉCTRICA

Salamanca,
del 3 al 5 de julio
de 1997

TOMO III

UNIVERSIDAD DE SALAMANCA
SECCIÓN DE INGENIERÍA ELÉCTRICA

UNIVERSIDADE NOVA DE LISBOA
D.T. DE INGENIERÍA ELECTROTECNICA