Sensor-Based Learning of Environment Model and Path Planning with a Nomad 200 Mobile Robot

Rui Araújo, and Aníbal T. de Almeida

Institute for 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: +351-39-35672; email: rui@isr.uc.pt

Abstract

This paper addresses the problem of learning sensor-based navigation of a mobile robot on an indoor environment, where the location, size, and shape of obstacles is assumed to be initially unknown to the robot. We use the parti-game multiresolution approach for simultaneous, learning of a world model, and learning to navigate from a start position to a goal region on the world. These two learning abilities are cooperating and enhancing each other in order to improve the overall system performance. It is assumed that the robot knows its own current world location. It is only additionally assumed that the mobile robot is able to perform sensor-based obstacle detection (not avoidance), and that it is able to perform straight-line motions. Results of experiments with a real Nomad 200 mobile robot will be presented.

1: Introduction

Path planning is one of the most important tasks for autonomous navigation of mobile robots. In this paper we consider the problem of autonomously navigating a mobile robot, without any external support or human intervention. It is assumed that there is no world map or model, initially available to the robot. Under this assumption, the robot path finding problem is considered. In this problem, a robot path avoiding collisions with obstacles must be generated, so that the robot moves from a starting location to a goal region in the world. An approach will be described where the mobile robot is able to concurrently construct a world model and learn a path to the goal.

Some approaches for mobile robot navigation are based on graph models. Such models are used for example in roadmap methods, where the world is represented as a network of curves describing the free-space regions. For example on the theoretical work [1], a retraction method based on the Voronoi diagram of the environment, is proposed for both model acquisition and navigation. In [2], a network of places is learned in a special environment exploration mode. The resulting graph is then used as the basis for navigation. Although network models offer elegant solutions, they require accurate sensor information, and are consequently difficult to implement in practice. On the other hand, grid-based models impose a constant resolution structure onto the environment without any selectivity concerning the nature and clutter of the world [3]. For overcoming this problem, some approaches have been based on a variable resolution state-space cell decomposition. For example [4] proposed a quad-tree based multiresolution partition method, and presented simulation results involving a 2-dimensional maze. In [5] it is presented a roadmap method, where free space is represented as a union of generalised cones, and [6] presents an approach that performs a cell decomposition of configuration space. In both these two methods, the A* algorithm [7] is used for searching an associated graph that is constructed into the model. Other geometric decomposition methods and road-map methods, as well as and potential field methods have been proposed (see [8] for an overview and further references). However those methods generally assume, and are only able to find paths, if a complete model of the environment is previously available (typically in the form of a pre-programmed list of polygons), and there is no learning or exploration capability. In [9], however, it is proposed a variable resolution approach where both path planning, and environment modelling are performed simultaneously. This approach also uses the quad-tree data structure, and simulation results are presented. Still other approaches don’t require a map at all, and do not construct a resulting world model. For example [10] proposes a reactive approach [11] that is based on a fuzzy system that learns to coordinate different behaviours, that must be predefined and programmed. In [12], a preprogrammed reactive system with some basic behaviours, guides an associated reinforcement learning system. This can be seen as an automatic teacher that enables convergence improvement on a system which otherwise would have greater learning difficulties.

In this paper we demonstrate the effectiveness of the parti-game learning approach [13] to the specific case of learning a mobile robot path from its initial position, to a known goal region in the world.
The organisation of the paper is as follows. Section 2 presents the learning controller architecture. Section 3 presents experimental navigation results with a real Nomad 200 mobile robot. Finally in section 4 we make some concluding remarks.

2: Controller architecture

The problem we wish to solve, may be stated as follows: The mobile robot is initially on some position on the environment, or world, and then it must learn a path to a known goal region in the world. The mobile robot controller architecture used in this work, is based on the application of the parti-game learning controller algorithm [13] to the specific case of learning a mobile robot path. It is a multiresolution approach that incorporates ideas from both graph-based, and grid-based methods. Spatial resolution is robustly chosen using a game-theoretic cell-splitting criterion. A database of previous experiences is incrementally maintained in real-time and used for planning. The algorithm does not have any initial internal representation, map, or model, of the world. In particular the system has no initial information regarding the location, the size, or the shape of obstacles. The mobile robot can simultaneously, learn a kind of map of its environment, and learn to navigate to the goal. These two learning abilities may be seen as cooperating and enhancing each other in order to improve the overall system performance. In applying the algorithm, we assume that the initial abilities of the mobile robot are only two. First, the robot is able to perform obstacle detection (not avoidance) operations using its own distance sensors. Second, the mobile robot is able to perform straight line movements between its current position and some other specified position in the world. This ability requires the knowledge of the robot current position and orientation. However, in this paper we do not deeply address the problem of mobile robot localisation. A simple method based on accumulation of encoder information is used for mobile robot localisation. Even though this simple approach induces errors, it was sufficient to validate the learning approach. The “straight-line mover” works as a local greedy controller, which can be used to move the system greedily towards a desired state. However, there is no guarantee that a request to the greedy controller will succeed (e.g. a movement may fail because of the presence of an obstacle that is detected to be obstructing the robot normal path).

2.1: The algorithm

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, …, N. In this paper we will assume that the cells are all axis aligned hyperrectangles. 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. In our case we define a two dimensional state-space, where the state vector, s = [X Y]T, is composed by the two position coordinates of the mobile robot (X and Y). Thus cells reduce to axis aligned rectangles. Real-valued states and cells are distinct entities. Each real-valued state belongs to (is an element of) one cell, and each cell contains a continuous set of real-valued states (i.e. a cell is a set of states). For example the right partitioning on figure 1 is composed of eleven cells, which are labelled with numbers 1…11. 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 (a geometric model).

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 towards 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

\[
\text{NEXT-PARTITION}(s, j) = \\
\{ \text{if we became stuck} \} \\
\{ \text{the cell containing the exit state otherwise} \}
\]

In our work the test for sticking performs an obstacle detection with the distance sensors of the mobile robot. In general, we may have \( \text{NEXT-PARTITION}(s, j) \neq j \), because the local greedy controller is not guaranteed to succeed. In fact, a cell-aim operation may fail be-
cause of either a detected obstructing obstacle, or by attaining a cell that was not the selected one. 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 the system starts “aiming.” The outcomes set of an action $j$ in cell $i$, is defined as the set of possible next cells:

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

When the system is on the real-valued state $s$ of a cell $i$, one of the key decisions that the algorithm has to take is to choose the cell at which the system should aim using the local greedy controller. The sequence of cells traversed to reach the goal cell, is planned using a game-like minimax shortest path approach. All next-cell outcomes, that are possible results of a certain cell-aim operation, may be viewed as “response-moves” available to an imaginary adversary that would be working against our objective of reaching the next cell, and ultimately reach the goal. The next cell on the path is chosen taking into account a worst case assumption, i.e., we imagine that for each cell we may aim, the adversary is able to place us on the worst position on the current cell such that the next cell that results from the aim is also the worst. In this way we always aim at the neighbouring cell with the best worst-outcome. In this framework, we can define the minimax 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 the intended next cell is $j$, an adversary is allowed to place us in the worst position within cell $i$ prior to the local controller being activated. Specifically, the $J_{WC}(i)$ shortest-path is defined as:

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

except,

$$J_{WC}(i) = \begin{cases} 1 & \text{if } i \text{ is a goal cell} \\ 0, \text{ if } i = \text{GOAL} \end{cases}$$

The $J_{WC}(i)$ values can be obtained by Dynamic Programming methods [14], or by a minimax algorithm [7]. Note that on a standard shortest path problem (that is usually solved by the A* or Dijkstra algorithms, or by Dynamic Programming), there is no associated $\text{OUTCOMES}(i, k)$ set. On such a problem, whenever we choose to advance to an adjacent node on the graph, that new node will be unconditionally attained.

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, equations (1) and (2), are 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 we can reach the neighbour that is aimed. Algorithm 1 (see figure 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 $\text{OUTCOMES}(i, j)$, and equations (1) and (2) are solved, to obtain the, possibly new, “minimax shortest path”. The $\text{OUTCOMES}(i, j)$ sets are an important part of the world model that is incrementally constructed from experience, and play an important role on the operation of the algorithm. 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 par-

**Figure 2: Algorithm 1.**

**Algorithm 1**

**REPEAT FOREVER**

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

   1.1 **IF** there exists some $k'$ for which $(i, j, k') \in D$ 
   THEN: $\text{OUTCOMES}(i, j) = \{k | (i, k, k') \in D\}$ 
   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 
   $$j := \arg\max_{j' \in \text{NEIGHS}(i) \cap \text{OUTCOMES}(i, j')} J_{WC}(k')$$

   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**
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 (AW/C = ∞).
   2.2 Let Q′ := The members of Q who have any non-losing neighbours.
   2.3 Let Q″ := Q′ and all non-losing neighbours 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 P := P + 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

3: Experimental results and discussion

The learning approach described in section 2 has been implemented and tested using a Nomad 200 mobile robot. In this section we will present an experimental example. The Nomad 200 is a three wheel robot with zero turning radius. The robot has 16 ultrasonic range sensors and 16 infrared range sensors equally spaced around its body, and a planar laser range finder. All these sensors are mounted on a turret that can rotate independently of the base. On the experiment here reported, the infrared range sensors were used to perform obstacle detection. Both the steering and the drive motors have encoders which provide the Cartesian location of the robot with a resolution of 2.54 mm. However the actual position accuracy may depend on other factors such as slippage between the robot’s wheels and the floor. The steering and turret angles may be controlled and measured with a resolution of (1/100) degrees.

On the presented example, the world environment is composed of some obstacles that make it difficult the task of learning to navigate from the starting position to the goal region. By using its infrared sensors, the robot was able to detect the obstacles (see figures 4, 5, 6, and 7). The experiment was organised as a sequence of trials to navigate from the start position to the goal region. The first trial starts with no model of the world. Subsequent trials start with and build
 enable a subsequent easy navigation to the goal. In fact, on the second trial (see figure 5), the system still needs to increase the resolution of the state-space partition, in areas the robot faces greater difficulties to navigate. The extensive exploration effort in this trial reflects the need to incrementally accumulate information about cell-aim outcomes regarding the newly created cells. On trial 3 we observe a much more direct navigation to the goal (see figure 6). However, in this trial some transitions between cells created on trial 2 are still attempted and the resulting outcome information is accumulated. On figure 7 it can be seen that, on trial 4, the mobile robot has already learned to promptly navigate to the goal.

The state-space partitioning is one of the key elements of the world model that is learned/constructed by the algorithm. It can clearly be seen that the robot increases the partition resolution on areas where it faces greater difficulties to navigate. On areas where navigation is easy, and on areas that the robot does not need to visit, the partition resolution is kept low. This enables a lower number of cells if comparing with some other cell-based approaches.

On figure 4, but mainly on figure 5, we observe that some obstacles are slanted when compared to figures 6 and 7. The reason for this is that, as already described, we do not deeply address the problem of mobile robot localisation. We simply used accumulation of encoder information to perform robot localisation. Even though this simple approach induces errors, it was sufficient for our learning experiments. Under those conditions, since the robot performs greater amounts of motion on the first two trials, the corre-
sponding localisation errors are also greater. Errors on robot self-localisation, are reflected on errors regarding the position where the it "thinks" the obstacles are. From the figures we can roughly say that the most important errors are on the orientation component. The location accumulators were set to correct values at the beginning of each trial. In spite of these localisation errors, the robot was still able to learn to navigate to the goal. However, it should be interesting to improve the method, in order to make it more robust to uncertainty in localisation.

In spite of being sufficient for its operation, the algorithm maintains limited amount of knowledge on the environment, that is kept in an indirect form of memorised outcome experiences. Also, some required knowledge forgetting is present on involved algorithm 2 (cf. step 2.6). This causes some of the search trajectories to be redundant (cf. figures 5 and 6). This could be mitigated through trajectory filtering by predictive on-line planification with a better world model, or by further improvements on the cell splitting strategy.

On the experiments conducted with the real robot, it was observed that the computational costs of the two algorithms that are used on the learning approach, are negligible when compared to the total time costs involved when making the mechanical robot evolve on its environment. In fact, some steps of the approach depend on the completion of mechanical motions of the robot which use a much greater proportion of time. For example on the first experiment that is presented in [15], on six trials on a “difficult” world, the ratio between the computational costs of the algorithms, and the total equivalent real-time elapsed (not including time used on robot pure rotations), is as low as 0.028. By this reason the computational costs are not a restriction to the practical implementation of the algorithm. It is believed that it is important for evaluating the performance of the approach, to include not only the computational costs, but also the world exploration and modelling efforts. Those ideas are used in an analysis conducted in [15].

4: 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. The robot constructs its own model of the world. The constructed model is simultaneously used as a basis to perform path planning. This enables learning to navigate the robot to the goal region. Results of experiments demonstrating the application of the learning approach to a real mobile robot were presented.

References


