Intelligent Control and Decision-Making demonstrated on a Simple Compass-Guided Robot

L. Seabra Lopes (*), **N. Lau** (*), **L.P. Reis** (**)
(*) Dep. Electrónica e Telecomunicações, Universidade de Aveiro, Portugal
(**) Lab. Inteligência Artificial e Ciências da Computação, Universidade do Porto, Portugal

Abstract: This paper presents the architecture and algorithms developed for Dom Dinis, a simple compass-guided robot built by the authors. This includes environment exploration, task planning and task execution. Environment exploration, based on repeating a reactive goal search, enables a progressive construction of a grid-based map. Based on the (possibly incomplete) map, the robot is able to plan its tasks. The execution capabilities of the robot include exception handling. Essential to all these capabilities is the knowledge of the robot's position in the world. The position is computed based on tracking traversed distances and followed orientations. Orientation is given by a compass. Dom Dinis doesn't use wheel encoders at all.

Keywords: basic behaviors, compass-based navigation, environment exploration, decision-making

1. Introduction

Integrating intelligence in robots will considerably enlarge the spectrum of robotics applications. In manufacturing, for instance, flexibility (namely in terms of product variety and batch size) is an increasingly desirable feature. This implies increasing the modularity and reconfigurability of the hardware and, therefore, reducing the structuration constraints. In the emerging field of service robotics, the absence of structuration constraints is the rule.

As the degree of structuredness and predictability of the environment decreases and the interaction with non-expert humans becomes more intense, robots will need a deeper understanding of what is going on around them. In some way, they will have to be intelligent.

Reactive approaches addressed with some success the problem of situated activity, coping with dynamically changing environments. In particular, the display of emergent intelligent-looking behavior by reactively controlled robots is considered one of the successes of the approach. However, despite the modular and incremental design that is often used, reactive systems seem to be difficult to build, requiring a great engineering effort to hard-code the desired functionality.

Being limited to applications requiring no explicit representation of the world is a major limitation of purely reactive systems. For instance, in mobile robots, any solution superior to random motion requires some representation of space.

A related fundamental limitation of this type of systems is that they are generally unable to display adequate goal directed behavior. On the other hand, when new situations arise, the ability to plan ahead could avoid time-consuming and possibly dangerous explorations of the environment. Moreover, a search in an internal model is orders of magnitude faster than the same search performed on the environment. There is a great difference between emergent clever-looking behavior and truly intelligent behavior. Research in *emergent mind* remains purely speculative (Havel, 1993). While purely reactive systems continue to be investigated, many researchers started to develop hybrid systems. Typically, reactivity is employed in low-level control and explicit reasoning is employed in higher-level deliberation (Georgeff and Lansky, 1987; Ambros-Ingerson and Steel, 1988; Arkin, 1989; Connell and Mahadevan, 1993; Firby *et al.*, 1995; Seabra Lopes, 1997).

The first robots developed by the behavior-based robotics community were essentially reactive robots. The absence of explicit representation and reasoning processes was seen as an advantage (Brooks, 1991). In recent years, the behavior-based approach as evolved in the direction of integrating more cognitive abilities (Arkin, 1998). Some of these abilities are implemented based on methods from classical artificial intelligence.

In this paper, we present the architecture and algorithms developed for a simple micro-mouse robot equipped with a compass. This robot, called Dom Dinis, was built by the authors for 1999's edition of *Micro-Rato*, the only robot competition regularly organized in Portugal. Dom Dinis ranked 2nd in a list of 41 robots initially registered. Additionally, Dom Dinis won the *Micro-Rato'99 Innovation Award*, sponsored by Microsoft (Portuguese branch).

The architecture of Dom Dinis integrates basic behaviors, environment exploration, task planning and task execution with exception handling. The tasks are, of course, very simple, but the demonstration of those functionalities on a real robot seems worth of report. The use of a compass for position computation and, therefore, for localization is another aspect to emphasize.

The paper is organized as follows: Section 2 presents the scenario of the Micro-Rato contest. Section 3 presents the hardware and block architecture of Dom Dinis. Section 4 presents the method for compass-based localization. Section 5 is devoted to environment exploration and map building. Sections 6 and 7 are devoted to task planning, execution and exception handling. Finally, section 8 concludes the paper.

2. Scenario and Tasks

The scenario for which Dom Dinis was designed is the scenario of the Micro-Rato contest. The robot's task is to go

0-7803-6583-6/00/$10.00 © 2000 IEEE

from a start area to a goal area in the shortest time possible and minimizing collisions with the environment or the other competing robots. Three robots compete simultaneously. The goal area is identified by a 940nm infra-red light source modulated at 25KHz placed at its centre. The competition area, a square of 5×5 m, is enclosed by walls. Inside this area, other walls/obstacles are placed forming a labyrinth. The minimum distance between every two obstacles is 0.5 m. All walls, obstacles and ground are dressed with light colour paper having a good infra-red light reflection coefficient. This is so in order to simplify obstacle avoidance based on IR sensors. The goal area is a 1m diameter circle dressed with a dark material of low infra-red reflection coefficient.

The kind of goal involved in this contest can be approached in two ways. The first one is the purely reactive approach. In this case, the robot follows the goal light until it finds an obstacle. When that happens, the robot tries to avoid the obstacle. When the obstacle is no longer visible, the robot follows the goal light again, and so on. The reactive approach basically conducts a real-time search for the solution. Depending on the complexity of the labyrinth, this approach may lead the robot to dead-ends and, eventually, to failure in reaching the goal.

The other approach is based on a task plan. In the case of the Micro-Rato contest, the task plan is simply a trajectory or path linking the start area to the goal area through the labyrinth. This plan or path can be either provided by a human or planned automatically based on a representation of the labyrinth. For Dom Dinis, the plan-based approach was followed. The algorithms that were developed rely on a compass for position calculation and movement orientation.

3. Anatomy of a Robot

Dom Dinis is a wheeled robot (Fig. 1). Part of the hardware was supplied by the organization of Micro-Rato'99. It is a round robot, with a diameter of 24 cm and a height of 26 cm. The control is done using a MPR11 board, which includes the 68HC11 micro-controller. This board must be linked to a memory board. In our case we used the ME11 board which includes 32KB of memory and also two motor drivers.

The obstacle detection is performed by three infra-red (IR) proximity sensors, each of them including an emitter and a receiver. These sensors allow, after calibration, the detection of objects up to a distance of 25cm. Objects beyond that distance may be mixed up with the light source signal that indicates the goal area or even with camcorders. Another IR sensor, rigidly mounted at the light source height and oriented frontwards, carried out the detection of the IR light signal coming from the goal area. A tube protects this IR receiver in order to minimize the influence of natural light or of other sources of IR light in the measured values. The ground sensor, responsible for goal area entrance detection, is an IR digital sensor, mounted at the bottom of the robot, checking ground reflexivity. Another board, IO_NOVA, developed at the Univ. of Aveiro, manages the interface between all these sensors and the control board.

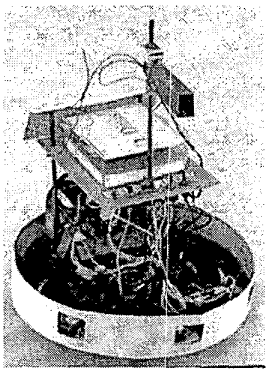


Fig. 1 - Front view of Dom Dinis

The main hardware innovation in Dom Dinis is the inclusion of an electronic compass. In nature, the senses of distance and orientation enable animals to have an idea of position. The integration of compasses in robots is a possibility that robot builders have been considering (Everett, 1995). In practice, however, very few robots reported in the literature are equipped with a compass. A recent and rather comprehensive survey about behavior-based robotics does not mention a single example of a robot equipped with a compass (Arkin, 1998).

The compass integrated in Dom Dinis is the Vector2X from Precision Navigation Inc.. This compass has a resolution of 1° and a precision of 2° RMS. It provides a SPI interface. The 68HC11 also includes an SPI interface but some adaptations were necessary. Orientations are retrieved from the electronic compass by a routine in the form of a finite-state automaton that implements the reading protocol. A timer, working at 1000Hz, triggers the execution of this routine. This provides approximately 5 orientation measures per second.

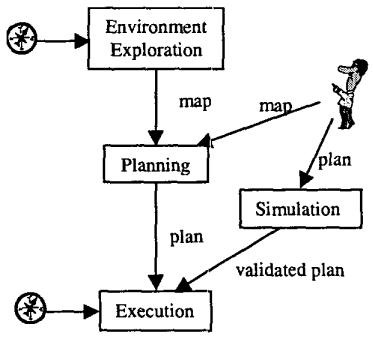


Fig. 2 - Functional architecture of Dom Dinis

Fig. 2 shows the functional architecture of Dom Dinis. Environment exploration allows to build a map. Based on the (possibly incomplete) map, the robot is able to plan its tasks. The execution capabilities of the robot include exception handling. Due to memory limitations, these

functional modules cannot be simultaneously downloaded to the robot. In any case, the functionality is demonstrated. In Micro-Rato, since there wasn't a training phase, the plan was directly provided to the robot.

4. Position Calculation and Calibration

In the control and decision-making algorithms of Dom Dinis, knowledge about the position of the robot is very important. The problem of localization is usually solved with help of encoders placed at the robot wheels. In our robot, the same result is obtained based on the compass. More precisely, Dom Dinis keeps track of location based on a sense of traversed distances and followed orientations, just like many animals do.

During motion, the robot performs regular updates to two global variables, x and y, that represent the robot's position in the world. This update is done based on the motion direction, θ, given by the compass, the speed, v, and the time elapsed since the last update, Δt. Speed and time allow to compute the distance, Δs. It is then trivial to compute the new values of the position variables:

$$\Delta s = v \times \Delta t$$
$$x_{i+1} = x_i + \Delta s \times \cos(\theta)$$
$$y_{i+1} = y_i + \Delta s \times \sin(\theta)$$

The value of the speed used in these computations is the result of an interpolation based on experiments performed for three nominal speed values (namely 25, 50 and 75) and several battery levels. It is, therefore, a 2D interpolation. For example, at an average battery level, the nominal speed of 75 (near to the maximum speed) translates into a real robot speed of 0.23 m/s.

Following this method, it is possible, in obstacle-free motion, to keep an accuracy level of 0.3 m for position (x,y). When the robot meets other competing robots or finds itself in dead-ends, position accuracy can degrade. For this reason, it is important that the robot can validate and re-calibrate the position variables.

In general, this problem can be solved by recognizing landmarks and calibrating the position relative to them. The environment of Dom Dinis, in the context of Micro-Rato, is not as rich as a natural environment would be. Nevertheless, it was possible to find ways of validating and calibrating the position variables. Validation is performed based on labyrinth dimensions. When the computed position is outside these dimensions then the new position becomes the position inside the labyrinth that is closer to the old position.

For calibration, there is in Micro-Rato's world another very good source: the goal light. On one hand, the emitted light signal allows to calculate the distance between the robot and the light emitter with an accuracy of 0.3 m (Fig. 3). On the other hand, the compass allows to determine the direction of the light emitter relative to the robot. Based on the position of the light emitter, the distance and the direction, it is possible to calculate an alternative value for the position, $P_l = (x_l, y_l)$.

Position calibration is performed based on a confidence measure, $\xi \in [0,1]$, that is updated during execution. This measure is increased when calibration is performed and decreased when the robot finds situations that typically reduce position accuracy. Finally, position calibration consists of re-defining the position, $P = (x,y)$, in the following way:

$$P = \xi \times P + (1-\xi) \times P_l$$

i.e., the new position is a weighted average between the current position and the position computed based on goal light distance and direction. The weight of the current position in this average is the confidence ξ.

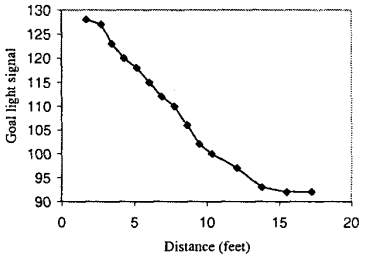


Fig. 3 – Goal light signal *versus* distance

5. Environment Exploration and Map Building

Maps can provide guidance beyond the horizon of immediate sensing (Arkin, 1998). A map is useful for resource location, avoiding nearly undetectable obstacles by remembering their locations, route selection based on relative distance or route selection based on going through places encountered in the past. Dom Dinis, in order to plan a task, uses a map of the labyrinth. This map is provided by the human user. In this section we propose a method for environment exploration and map building that was only partially implemented.

The representation used for the map is based on a grid. Since the error associated to the position is 0.3 m, the resolution of the grid is also set to 0.3 m. This means that, for the labyrinth of Micro-Rato (5m ×5m), the map will be represented by a grid with 17 rows and 17 columns.

Each cell in the grid can be in one of the following three states:

- free space
- obstacle or wall
- unknown (i.e. not yet explored)

At the beginning, the robot only knows the exact coordinates of both its initial position and the position of the goal light. All grid cells are marked as 'unknown'.

Environment exploration is not a systematic scanning procedure for building a complete map. Rather it is based on repeating a reactive search process that leads the robot from a given initial position to the goal area (fig. 4).

This search is a composition of some of the basic behaviors that were implemented. All of these behaviors use the information provided by the compass.

```
repeat
    search for direction of goal light → dir
    move in the direction dir
    if wall found
      then follow wall until returning to dir
until in goal area
```

Fig. 4 – Reactive search for the goal area

The reactive search algorithm takes decisions based on a minimal set of local sensor information, namely: the direction of the goal light (light sensor), the current direction (compass) and the presence of obstacles (proximity sensors).

In open space the algorithm drives the robot in the direction that leads directly to the goal area. When an obstacle is found, the direction of the goal light at that moment is saved, and the robot tries to surpass the obstacle following its contour. The problem of "When should the robot stop going around the obstacle?" was solved by observing that any trajectory leading to the goal area must cross the saved direction while going around the object. We thought, and it has been verified, that the point where the robot reaches the saved direction is a good one for reevaluating the situation and re-initializing the algorithm.

This type of reasoning is able to surpass a large variety of obstacles, however if the robot tries to go around the obstacle clockwise and the nearest passage is in the opposite direction (or vice-versa) it can loose too much time. There are also some odd obstacles for which the algorithm makes the robot follow a loop. To avoid these situations if the robot doesn't reach the saved direction after a random (adjustable) duration it will also reinitialize the algorithm.

While search proceeds, the robot updates the map grid cells as 'free space' or 'obstacle/wall'. This search can be performed avoiding history, meaning that the robot prefers to move through unknown space. Already explored space is only traversed a second time when there is no alternative. In this way, the robot not only explores a greater portion of the environment but it is also able to handle dead-ends.

Fig. 5a shows the map built by the robot during one first run of the reactive search for the goal. Fig. 5b shows the map updated after a second run. Fig. 5c shows a complete map of the labyrinth (this is actually the labyrinth of *Micro-Rato'99* - an easy one). In general, the free space found after the first explorations is precisely the free space that matters in order for the robot to solve its tasks.

6. Planning

Once a map is available, the robot can plan its tasks, i.e., the robot can determine (near-)optimal paths in the labyrinth.

The approach is based on a segmentation of the map into square regions. This segmentation is essential in order to keep the computational complexity of the planning process within reasonable bounds. The segmentation algorithm is given in Fig. 6. Fig. 7a presents the result of the application of the algorithm to the map of Fig. 5c. The goal light is marked as a regular obstacle (in the center in the case of figs. 7a-c). Note that map segmentation and path planning can be performed even when the map is incomplete, i.e., when some cells in the map remain 'unknown'. In this case, what must be done is to treat 'unknown' cells during segmentation just like obstacles. From the segmentation a topological map or connectivity graph is derived.

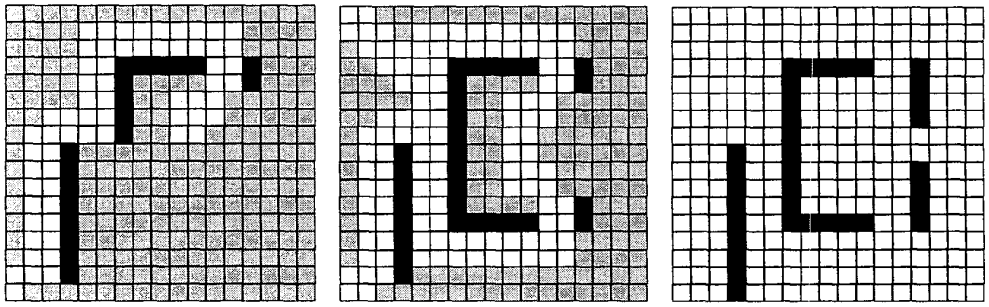
```
for i:=1 to Nrow do
 for j:=1 to Ncol do
   if cell (i,j) is not yet included in a
      previously created square region
     then determine and create the largest
          square region composed of only free
          space cells and having cell (i,j) as
          its top-left vertex
```

Fig. 6 – Map segmentation algorithm

Based on the connectiviy graph, a plan can finally be determined by conducting a search process. A*-search was the algorithm that was used (see for instance Russell and Norvig, 1995). The resulting plan is a sequence of regions that must be traversed in order to go from a given initial position to a given goal position.

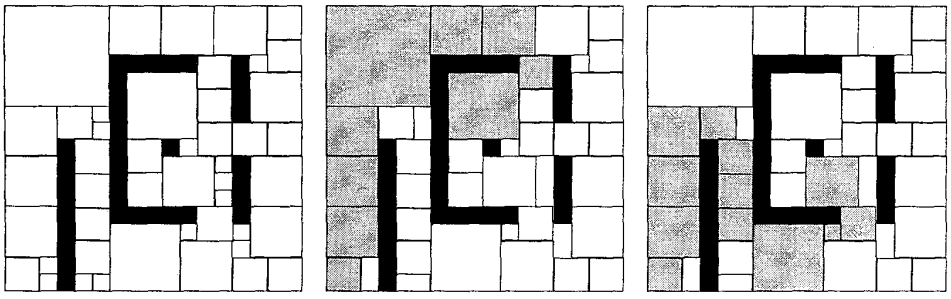
A*-search is conducted in a problem graph, meaning that redundant search paths are excluded. For this type of search, two cost functions must be defined:

- $g(s)$ - the real cost of going from the initial state to the current state s.
- $h(s)$ - an estimate of the cost of going from the current state s to a solution state.



a b c

Fig. 5 - Three phases of environment exploration (example)



a b c

Fig. 7 - Map segmentation and two alternative plans

The potential of reaching a solution state via state s is evaluated by the estimated total cost given by:

$$f(s) = g(s) + h(s)$$

In the planning application of this paper, a state s corresponds to a region of the map. $g(s)$ is defined as the length of the path linking the geometric centers of the traversed regions. $h(s)$ is the straight-line distance to the goal position. Since $h(s)$ is an optimistic estimate (never overestimates the real cost), it is then guaranteed that the first plan that is encountered is the optimal plan with respect to the real cost function $g(\,)$.

Note that $g(\,)$ is the cost of following a path that links the centers of the traversed regions. Of course, this is not the optimal path through those regions. Therefore, the obtained plan is optimal with respect to the $g(\,)$ function, but is not necessarily optimal in absolute terms. In any case, it will be near-optimal.

Figs. 7b and 7c show two plans for going from the start position to the goal area, located in the center of the map. Dom Dinis can perform all these planning computations (segmentation and search) in only a few seconds (typically less than 5s).

We suppose that the map segmentation method presented above is original. The resulting map regions are square, instead of arbitrarily rectangular, for two reasons. On one hand, a path through the centers of connected regions is always feasible. On the other hand, arbitrarily rectangular regions would complicate the definition of the $g()$ cost measure. A good example of this is a corridor. Suppose that the robot is in one end of a corridor and that the robot's task involves crossing a doorway somewhere in that corridor. If the corridor is segmented as a long rectangular region and the cost is calculated based on the distance to the center of that region, then moving from the corridor region to the doorway region will be "cheaper" for doorways in the center of the corridor.

The presented segmentation method is computationally efficient, produces "natural" map regions and enables a dramatic improvement in planning efficiency with respect to the base grid.

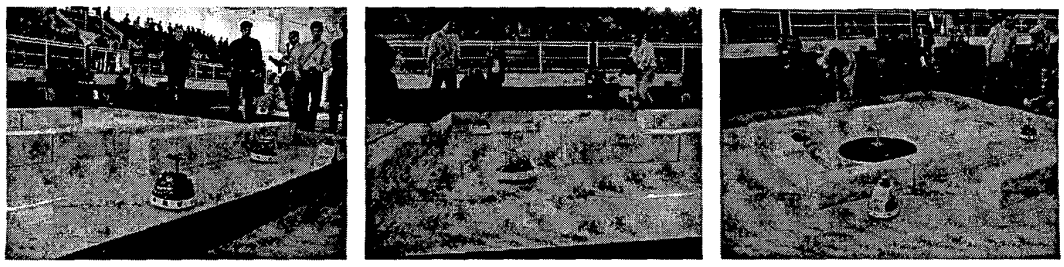
The application of A*-search to path planning in behavior-based robots has been reported before (Arkin, 1989; Simmons and Koening, 1995). The application of more elaborate Artificial Intelligence planning techniques to robotics has also been reported (Goel et al., 1994; Haigh and Veloso, 1996).

7. Plan Execution and Exception Handling

As we have just seen, Dom Dinis' plans are sequences of connected regions. From this sequence, a more precise trajectory, defined by a sequence of points, can be derived. The most obvious option (although not optimal) is to define the trajectory by the geometric centers of those regions (as mentioned in the previous section, this is also the base of cost computations during planning).

Between two consecutive points in the path, the robot executes the `guided_motion()` behavior, that leads the robot from its initial position to a given end position. If there are no obstacles in the way (other robots, for instance), the robot's trajectory approaches the straight line. In any case, position update is always based on the direction actually followed.

This behavior is based on a normal sensing-and-action loop with proportional control. In obstacle-free motion, the robot moves at top speed. In order to keep it permanently



a b c

Fig. 8 - Three moments of the competition

oriented towards the end-point, a correction proportional to the difference between the current and the target directions is applied to the wheel speeds.

When the robot finds an obstacle, it reduces speed and stops applying the speed correction just mentioned. At this point, a reactive attitude is adopted in order to more easily surpass the obstacle. When the encountered obstacles are other robots (or of comparable size), Dom Dinis is able to surpass them very easily and efficiently and without significant degradation in position accuracy. Surpassed the obstacle, the robot resumes the default linear behavior.

If the obstacles found by the robot are actually walls, the problem is more serious. This typically happens when, after accumulating position errors, the robot fails to cross a narrow passage. If the robot starts to move in the wrong direction, the "avoid obstacles" strategy will most likely take it far away from its trajectory and, eventually, make it enter dead-ends from which it may be hard to escape.

In order to handle these undesirable situations (execution exceptions) two heuristics have been developed. One of them prevents the robot from going more than 0.6 m far from the ideal trajectory (i.e., the straight line linking the initial and end points). When this limit is reached, the robot rotates to the direction orthogonal to the ideal trajectory. Probably, it will still see the obstacle, but now it will go around in the opposite direction. The other heuristic is based on decomposing the trajectory between two points into several segments. This heuristic is aimed at forcing the robot to follow the ideal trajectory as closely as possible and, in particular, at avoiding dead-ends.

Fig. 8 shows three moments of the competition, that illustrate the advantage of having a plan. In 7a, one robot with good wheel motors, departs very fast leaving Dom Dinis and another robot behind. In 7b, this trend is strengthened. Finally, in 7c, Dom Dinis is approaching the goal area, while the first robot is already finishing a complete tour to the labyrinth.

Fig. 9 shows the final, in which Dom Dinis (2nd) and the winner approach the goal almost at the same time.

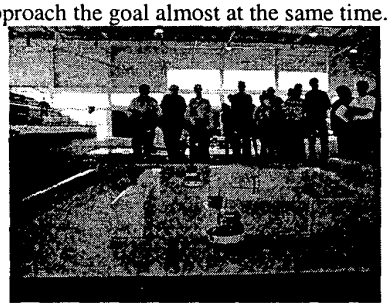


Fig. 9 - One moment of Micro-Rato's final round

8. Conclusions

A truly useful intelligent robot should be able to explore its environment, build a map of it, plan its tasks and execute them handling eventual exceptions. Dom Dinis is a real robot that performs these activities in a simplified environment. Dom Dinis was built for a micro-mouse contest where it won an innovation prize. At the hardware level, the main innovation of Dom Dinis is the inclusion of a compass. Compasses have seldom been used in intelligent / behavior-based robots. With this simple robot we demonstrated that the compass can be used, not only for guiding the robot, but also for calculating the robot's position. Dom Dinis does not have encoders in the wheels. Environment exploration is based on a reactive compass-guided search process for the robot's goal. The result is a grid-based map. The planner segments the map into "natural regions", builds a connectivity graph and finally searches for the best path. Dom Dinis is able to execute these plans even in the presence of other robots.

References

Ambros-Ingerson, J. and S. Steel (1988) Integrating Planning, Execution and Monitoring, *Proc. of the 7th National Conference on Artificial Intelligence (AAAI-88)*, St. Paul, Minnesota, pp. 735-740.

Arkin, R.C. (1989) Navigational Path Planning for a Vision-based Mobile Robot, *Robotica*, vol. 7, pp. 49-63.

Arkin, R.C. (1998) *Behavior-Based Robotics*, The MIT Press.

Brooks, R.A. (1991) Intelligence without Reason, *12th International Joint Conference on Artificial Intelligence* (IJCAI-91), Sydney, Australia, pp. 569-595.

Connell, J.H. and S. Mahadevan (ed.) (1993) *Robot Learning*, Kluwer Academic Publ., Boston.

Everett, H.R. (1995) *Sensors for Mobile Robots: Theory and Applications*, A.K. Peters Ltd.

Firby, R.J., R.E. Kahn, P.N. Prokopowicz and M.J. Swain (1995) Collecting Trash: a Test of Purposive Vision, *Proc. of the Workshop on Vision for Robots*, Pittsburgh, Pennsylvania, pp. 18-27.

Georgeff, M.P. and A.L. Lansky (1987) Reactive Reasoning and Planning, *Proc. 6th National Conference on Artificial Intelligence* (AAAI-87), Seattle, p. 677-682.

Goel, A., K. Ali, M. Donnellan, A. Gomez de Silva Garza, and T. Callantine (1994) Multi-strategy Adaptive Path Planning", *IEEE Expert*, vol. 9, pp. 57-65.

Haigh, K.Z. and M. Veloso (1996) Interleaving Planning and Robot Execution for Asynchronous User Requests, *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS'96), Osaka, Japan, pp. 148-155.

Havel, I.M. (1993) Artificial Thought and Emergent Mind, *Proc. Int. Joint Conference on Artificial Intelligence*, Chamberri, France, 758-756.

Russell, S.J. and P. Norvig (1995) *Artificial Intelligence. A Modern Approach*, Prentice Hall, New Jersey.

Seabra Lopes, L. (1997) *Robot Learning at the Task Level: A Study in the Assembly Domain*, Universidade Nova de Lisboa, Ph.D. Thesis.

Simmons, R. and S. Koening (1995) Probabilistic Robot Navigation in Partially Observable Environments, *Proc. Int. Joint Conf. Artificial Intelligence* (IJCAI-95), Montreal, Canada, p. 1080-87.