Path Planning and Navigation of Mobile Robots in Unknown Environments

Torvald Ersson and Xiaoming Hu
Optimization and Systems Theory / Centre for Autonomous Systems
Royal Institute of Technology, SE 100 44 Stockholm Sweden
(ersson@math.kth.se, hu@math.kth.se)

Abstract

For a robot trying to reach places in at least partially unknown environments there is often a need to replan paths online based on information extracted from the surroundings. In this paper is is assumed that the sensing range of the robot is short compared to the length of the paths it plans and the environment is modeled as a graph consisting of nodes and arcs. The replanning problem is solved using the network simplex method . The applicability of the planner is demonstrated by integrating it with a navigation control strategy. Simulation results show that the approach works well.

1 Introduction

Path planning and navigation for mobile robots, in particular the case where the environment is known, is a well studied problem, see, for example, the book by Latombe [4] and the references therein. In practice, however, one problem is that often no complete knowledge about the environment is available. Having a detailed map with all the obstacles marked seems to be unrealistic for most situations. In many outdoor applications the robots can determine their coordinates by using, for example, GPS. However the knowledge about the surroundings may often be very limited. Under such conditions there is too much uncertainty for a very detailed plan to make sense. For preplanning purposes a coarser choice is probably good enough. Additionally it is important to be able to replan the path online based on new information obtained by sensors while navigating.

A natural way of updating plans is to first select a path based on the present knowledge, then move along that path for a short time while collecting new information. Based on the new findings the path is then replanned. This methodology is often used in the literature for path planning in unknown areas. One of the original motivations for studying this problem was the terrain acquisition problem, where a robot is required to produce a complete map of an unknown terrain. In many publications graph methods are used for solving the task. For example, [6] considers disjoint convex polygonal obstacles and presents proofs on convergence. [5] does not use any constraint on the shape of the obstacles and tries to minimize the length of the path during the operation. In [7] results for more general graphs are presented.

In this paper an online path planning algorithm, based on the so-called network simplex method, is proposed. Compared with the aforementioned graph methods, the information stored about the environment is strictly intended for path planning and less details about the obstacles are needed. Although the algorithm of this paper to some extent is similar to the $D^\star$ algorithm [8, 9], the two strategies differ in a number of ways. Firstly, here a different discretization scheme is used. Instead of modeling the environment as a set of squares this paper uses a graph representation. Secondly, while producing a path from the start to the goal $D^\star$ has to solve the shortest path problem for every possible starting point. The algorithm of this paper does not require these extra solutions. Here the shortest path problem is thought of as a minimum cost flow problem and solved by the network simplex method. Furthermore, in this paper the feasibility of the algorithm is justified by integrating it with a generic navigation control strategy, since the topic here is *online* path planning.

It is worth to mention that as a contrast a completely different online path planning approach is presented in [11]. There a path is computed using steepest gradient descent on an updated harmonic function.

It should be emphasized that in this paper the range of the sensors is assumed to be limited. A requirement for the method to work is that the sensing range is longer than the length of the longest arcs of the graph. I.e. the lower the sensing range the closer the nodes need to be

placed. In [10] the sensing constraint is relaxed and an extension of [8, 9] with special box patterns for sparse environments is presented.

This paper is organized as follows. First the path planning approach, using network simplex, is presented. A more general description of the planning approach is followed by an example where parameter values have been chosen. Then a navigation controller is described and integrated with the path planning. Finally simulation results illustrate the performance of the combined strategy.

2 Path Planning

Three basic assumptions used in this approach are:
• The robot has a short sensing range compared to the size of the region of interest.
• It senses radially from its position. I.e. obstacles can block the sensing in some directions.
• It knows its coordinates and orientation (for example, via GPS).

2.1 Discretization

For our modeling it is necessary that the local region being considered is discretized. Discretizing the problem obviously excludes lots of solutions. However in many cases no exact path following is needed and a coarse plan is satisfactory. Another concern might be that in some cases the planning can result in a non-smooth path. Fortunately this matter can be dealt with using a control algorithm that smoothens the motion.

In this paper the terrain is described by nodes connected by arcs. Each arc has a specified cost for moving along it. Another straight forward approach, used by [8], is to split the environment into squares/rectangles of equal size. Then one assigns a cost for a transition between two neighboring squares.

A similar description of the environment is to first place nodes in a grid pattern, then introduce arcs between neighboring nodes and assign costs to these arcs. (Naturally the positions of the nodes need to be known.) Two obvious representations are shown in figure 1 a) and b). (An arc drawn between two nodes in the pictures should be interpreted as two arcs, one in each direction.) The two patterns in figure 1 a) and b) are, as can be seen in figure 2, almost equivalent to the approach using squares in a regular pattern. The node-arc representation can however easily be extended to more advanced patterns such as the one in figure 1 c). This structure allows more planning options and makes shorter and smoother

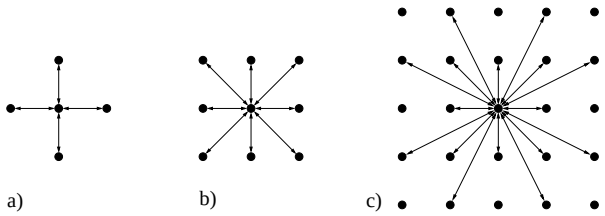

Figure 1: *a) "4-star" b) "8-star" c) "16-star".*

paths possible. Note that some of the arcs would correspond to moving between non neighboring squares. For

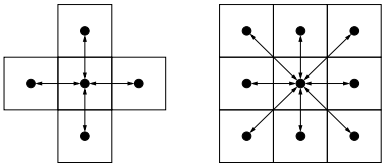

Figure 2: *"4-star" and "8-star" mapped onto a square pattern.*

a node arc approach to be interesting one, as previously mentioned, needs to have a sensing range that is longer than the length of the longest arcs. Apart from that restriction any kind of node arc network can be used.

The cost of an arc is naturally dependent on the distance between the nodes, but other aspects need to be taken into consideration as well. It is easy to include knowledge about the environment as e.g. lower costs of arcs in "friendly" terrain and very high costs for arcs affected by obstacles such as rocks, steep descents, creeks, fences, walls etc.. Of course this very high cost is ideally infinity, but in a world of computers infinity is replaced by a value much greater than the costs of the arcs that can be used.

This scheme for discretizing the surroundings makes it really easy to use the well known network simplex method for the path planning.

2.2 Minimum Cost Flow

For the sake of completeness this linear optimization problem is described.

Consider a network consisting of n nodes plus arcs connecting them. The unit cost for using the arc going from node i to node j is c_{ij}. The flow going that way is denoted x_{ij}. At node i there can be a flow supply or demand denoted b_i (supply:$b_i > 0$,demand:$b_i < 0$).

The problem is balanced if $\sum_{i=1}^{n} b_i = 0$. For every node inflow minus outflow equals demand/supply. The problem of minimizing the total flow cost of this network can be stated as the following linear optimization problem:

$$minimize \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1}$$

$$s.t. \quad \sum_{i=1}^{n} x_{ij} - \sum_{i=1}^{n} x_{ki} = b_i, \quad i = 1..n \tag{2}$$

$$x_{ki} \geq 0 \tag{3}$$

Only node-pairs (ij) corresponding to arcs need to be considered.

Network simplex [2] is a well known efficient method for solving such a problem. It works with so called basic solutions of the problem. Such a solution corresponds to a spanning tree. (A graph of arcs that connects all nodes of the network and does not include cycles.) The three steps of the method can be summarized as follows:

Step1. Start with a given basic feasible solution. Such a solution can be created by constructing a spanning tree.

Step 2. Compute node prices λ_i for each node i. This is done by solving

$$\lambda_i - \lambda_j = c_{ij} \tag{4}$$

for each i, j corresponding to a basic arc. For the above equation to have a unique solution one node price has to be assigned a value, e.g. $\lambda_n = 0$. The reduced costs r_{ij} for non basic arcs are computed using

$$r_{ij} = c_{ij} - (\lambda_i - \lambda_j). \tag{5}$$

If all r_{ij} are nonnegative the solution is optimal. Otherwise go to step 3.

Step 3. A new spanning tree is created by adding an arc to form a cycle and then eliminating another arc from the cycle. Select a non basic arc with negative reduced cost r_{ij} to enter the basis. Addition of this arc to the spanning tree will produce a cycle. Introduce a flow of value θ around this cycle. As θ is increased some old basic flows will decrease. θ is chosen as the smallest value that makes the flow in a basic arc zero. That zeroed arc now goes out of the basis. The new solution obtained in this step is then plugged into step 2.

The problem described above is more general than a shortest path problem. To model a path starting at node k and ending at node l a supply of one is placed at k, $b_k = 1$, and a demand of one at l, $b_l = -1$. All other b_i are equal to zero. in this case a basic solution includes the actual path flow represented by arcs carrying a flow of one, as well as arcs with zero flow. The latter ones are needed for creating a spanning tree.

Assume that network simplex has produced an optimal solution. What happens if a few costs c_{ij} are changed ? The solution is still both basic and feasible so step 2 is entered. If c_{ij} is changed for a basic arc, there might be a need to recompute all node prices before checking a large number of reduced costs of the network. If no cost of any basic arc is changed only the r_{ij} corresponding to the updated non basic arcs need to be checked.

Network simplex is not the fastest possible choice for solving a shortest path problem from scratch. However when having an optimal solution and changing a few costs it is quite efficient.

2.3 Path Planning Algorithm

The general idea is that a path is planned based on what is known right now. When the robot gets new information it considers choosing another path. Gradual learning about the surroundings results in better and better plans. Information about the environment is translated to arc costs c_{ij}. If the terrain is completely unknown the arcs can initially be assigned costs as if they were not affected by obstacles. Otherwise information from e.g. maps can be incorporated as suitable arc costs.

Step 1. Planning
Based on the present knowledge a path from the current node to the goal node is planned using network simplex. If the cost of this path is so high it is clear that an arc with 'infinite' cost is used the algorithm must be terminated. This means no feasible path exists within the network. If the cost is reasonable carry on to step 2.

Step 2. Moving and sensing
The platform moves along the first arc that is part of the solution while collecting information about the environment. At the end of the arc a new node is reached. If that node is the goal node the task is completed, otherwise step 3 follows.

Step 3. Reposing the problem
Unless the arc has been used before and nothing has changed the system now knows more about the surroundings and has updated a number of arc costs c_{ij}. This new information can be used for replanning. In order for the new planning problem to be correctly posed

the supply of one is moved from the previous node to the current one. The previous base solution is slightly modified to be a starting guess to the new problem. The only adjustment needed is setting the flow of the latest arc used to zero. In case no new information has been added this is an optimal solution. The algorithm now returns to step 1.

2.4 Some Practical Issues of Online Path Planning

Sometimes there are several paths with the same total cost, as in the case of a flat area almost free of obstacles. In these cases the proposed method will choose one of them arbitrarily. For some vehicles the choice does not matter that much, but for others it can make a big difference. Many mobile platforms, especially those in outdoor applications, are nonholonomic. One implication of this is that the range of steering is limited. Even if that range is sufficient a more winding path with unnecessary many turns is often unwanted.

Therefore ways of avoiding such choices are of interest. To prevent the path planner from choosing jerky paths one should lower the cost of going straight by a small quantity δ. This means that if a solution starting with a turn has the same cost as one starting by going straight before anything is adjusted, the δ will make the method pick the arc going straight. (figure 3) Once the robot has left the node the arc going straight will regain the cost it had before δ was subtracted. δ should be significantly smaller than the cost of an arc. It should be mentioned that in general an introduction of bigger temporary cost reductions can cause deadlock problems. In order to show the applicability of the approach one

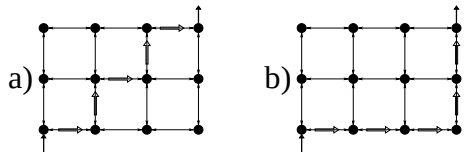

Figure 3: *a) If all the arcs have equal cost this winding solution is an optimal one. b) If there in every iteration is a slightly lower cost for going straight, this is the solution chosen when the first action is a step to the right.*

has to answer the following question: what costs can be estimated during motion? If the sensing range is only marginally longer than the length of the longest arcs it implies that:

• The robot can estimate the costs of the arcs that intersect the path.

• It can estimate the costs of the arcs starting and ending at the nodes it visits.

In case of a "4-star" pattern there are no intersecting arcs. For the "8-star" pattern the only arc intersection occuring is between diagonal arcs. If a "16-star" pattern is considered the situation is much more complicated. When this node-arc model is used there are three different arc types. In figure 4, it is shown how other arcs intersect the three in the interior of the grid. (Close to the borders the situation is slightly revised.) These cutting arcs will be visible when moving along the indicated types.

From a logical point of view often even more arcs can be measured. Consider a visible arc that is not blocked by obstacles. Assume that one of its neighbor arcs is parallel to it and lies within the range of the robot sensors. Then they both lie on the same line and within the sensing range. In that case it is possible to estimate the cost of that more distant arc as well.

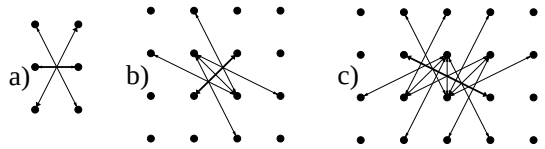

Figure 4: *Arcs of a) type 1, b) type 2 and c) type 3 are shown as thick arrows. Intersecting arcs are drawn as thin arrows.*

One could of course choose a denser grid. In such a case the sensing range may widely exceed the length of the longest arcs of the network. That means that more arcs are visible from the nodes and arcs used by the mobile platform.

2.5 Some Drawbacks

Discretizing a continuous problem is often a good way of making it easier, but doing so often introduces unwanted side effects. There are some obvious drawbacks with this graph approach. Firstly the solutions obtained are optimal for the discrete network problem, but not for the original problem. In figure 5 a) this is demonstrated. Imagine that the terrain is flat and free of obstacles. The real optimum is shown as a dashed line and the network optimum as white arrows.

Secondly in some cases an unfortunate choice of network might cause the discretized problem to be infeasible, even though the original problem has a solution (figure 5 b)). Local online modification of the network is one possible way of dealing with such situations.

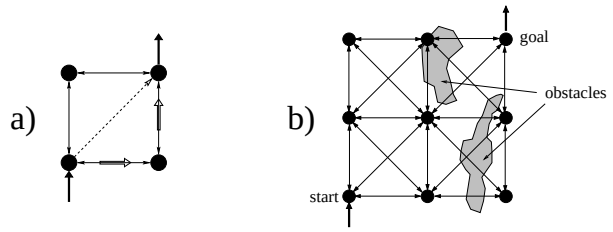

Figure 5: *a) The optimal solution for the discretized problem is clearly different than the optimal one for the original problem. b) The original problem is feasible, but the discretized is not.*

2.6 Maze Example

If the environment is fairly uncomplicated a robot without memory or planning skills could still have a satisfactory performance. To show the potential of the proposed planner a tough example with lots of possible deadlocks was designed. The maze-like environment chosen would cause many less sophisticated approaches to fail. As seen in figure 6 the planner did not have a problem finding a way through in a rational manner. When the example was tested on a computer with a 360 MHz UltraSPARC II processor the average planning task at a node took about 70 ms to finish. A grid consisting of 50x50 nodes connected with "16-stars" was used. The distance from a node to its closest neighbors was 1 length unit and the length of the longest arcs was 2.23. The sensing range used was 2.3 length units and δ was set to 0.1. The area was initially assumed to be obstacle free. For obstacle free arcs the costs were equal to the length of the arcs. Arcs blocked by walls were assigned a cost 10000 times higher than their length.

3 Integration of Online Path Planning and Control

Good and robust path tracking control strategies are needed in many applications and this has been a well studied topic. Naturally, how good a path following control is depends a lot on how well the path is planned. Many path planning strategies produce a smooth path where the curvature, for example, is minimized. The path planning algorithm of this paper, which is designed for navigation in an unknown environment, is quite different. Since a path produced by the algorithm is not even smooth, it is sometimes difficult to follow it exactly.

However, when the mobile robot navigates in an at least partially unknown environment, where online path planning is almost unavoidable, following the planned path exactly is not the top priority. The important thing is

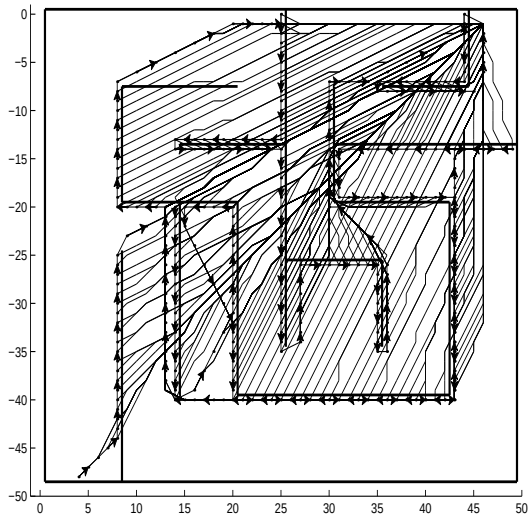

Figure 6: *Solution of the maze example. The dots and arrows show how the robot moves. Walls are drawn as thick lines and the thin lines are temporary plans on how to reach the goal in the top right corner. The starting point is situated in the bottom left corner.*

that the robot follows the path reasonably well in a very robust way. The robustness requirement is easily understandable since small hinders and external disturbances should always be taken into consideration, especially in outdoor applications.

In this section, by integrating the path planning algorithm with the so called virtual vehicle control strategy [3] , it is shown that the algorithm serves well for the purpose of navigating in an unknown environment. Of course many other control approaches could be interesting as alternatives. However here the aim is just to show that the planner can be used in practice.

3.1 The Virtual Vehicle Approach

For the sake of simplicity, only paths in a 2D-plane are considered. Suppose that a planned reference path is parameterized as:

$$x_d = p(s), \qquad y_d = q(s), \qquad 0 \le s \le s_f \tag{6}$$

where subindex d stands for desired. Consider a vehicle and a reference as shown in figure 7. Now assume that $p'(s)^2 + q'(s)^2 \neq 0$ and let

$$\begin{array}{ll} \Delta x = x_d - x, & \Delta y = y_d - y, \quad v = \sqrt{\dot{x}^2 + \dot{y}^2} \\ \Delta\psi = \psi_d - \psi, & \rho = \sqrt{(\Delta x)^2 + (\Delta y)^2} \end{array} \tag{7}$$

where v is of course the vehicle speed and (x, y) a reference point on the mobile platform, for example, the

mid point of the vehicle's front axle for a car-like robot. ψ denotes the orientation of the platform.

In the virtual vehicle approach the parameter s is treated as a function of the time $s(t)$. Its dynamics is governed by a differential equation that contains the tracking error ρ. Thus $s(t)$ is called the virtual vehicle (that the actual one should track). Suppose that one

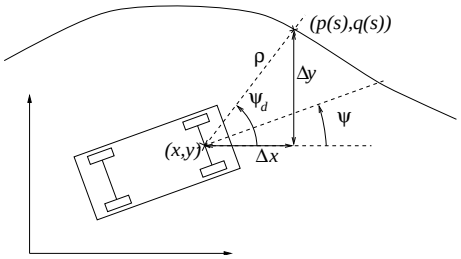

Figure 7: *Virtual vehicle geometry.*

can control the rotational and translational velocity of the platform. (They can be viewed as higher-level controls. For platforms that do not have direct control over the velocities, one needs to design the actuator control so that these velocity controls are realized.) Then the following generic control algorithm is given by [3]:

$$\begin{cases} \dot{s} = \frac{ce^{-\alpha\rho}v_0}{\sqrt{p'(s)^2+q'(s)^2}} \\ \omega = k\Delta\psi + \dot{\psi}_d \\ v = \gamma\rho cos(\Delta\psi) \end{cases} \tag{8}$$

where $\omega = \dot{\psi}$ is the rotational velocity and α, c, γ and k are some positive constants. As can be seen in the translational velocity control law, reverse drive is used when the magnitude of the deviation from the desired direction is too big.

In [3] it is shown that this control strategy is stable and quite robust for the two different platforms discussed there. Furthermore, the tracking error can be tuned arbitrarily small asymptotically but remains positive (note if $\rho = 0$ then ψ_d is ill-defined).

Since the position and orientation of our platform are assumed to be known the above closed-loop control approach can easily be applied here.

3.2 Path Planning Online Combined with the Virtual Vehicle Approach

It is obvious that in order to use the virtual vehicle approach, the path has to be smooth (C^1). Therefore there is a need to somewhat modify the path produced by the planning algorithm.

In fact, if relaxing the problem slightly one can say that a node is reached if the robot gets within a certain distance from it. This will define a "goal disc" centered at the node point. Between such discs trying to track a straight line along an arc seems reasonable. The radius of the goal area is limited by the fact that one needs to be close enough to evaluate outgoing arcs from the node. Otherwise the planning would be crippled by a lack of necessary information.

From an initial position (usually at the border of a goal disc) the vehicle will try to track a straight line leading to the next node area using the virtual vehicle approach presented above. When the robot gets there and planning has taken place the next node along the path is determined and the same procedure takes place again. This will go on until the final goal is reached. (figure 8)

If the starting node has position $\bar{r}_1 = [x_1, y_1]^T$ and the next node is located at $\bar{r}_2 = [x_2, y_2]^T$ the line between them can be described by $\bar{r} = \bar{r}_1 + (\bar{r}_2 - \bar{r}_1)\frac{s}{s_f}$, yielding

$$\begin{cases} p(s) = x_1 + (x_2 - x_1)\frac{s}{s_f} \\ q(s) = y_1 + (y_2 - y_1)\frac{s}{s_f} \end{cases} \tag{9}$$

and

$$p'(s) = \frac{x_2 - x_1}{s_f}, \qquad q'(s) = \frac{y_2 - y_1}{s_f}. \tag{10}$$

Clearly $p'(s)^2 + q'(s)^2 \neq 0$ unless $r_1 = r_2$. The reference could have s starting at zero or at a relatively low value. In simulation it turns out that the vehicle chases the reference point like a dog would chase a rabbit, rather than being in front or beside it. Since the robot chases the reference point, the reference line might extend beyond the next node point ($s > s_f$).

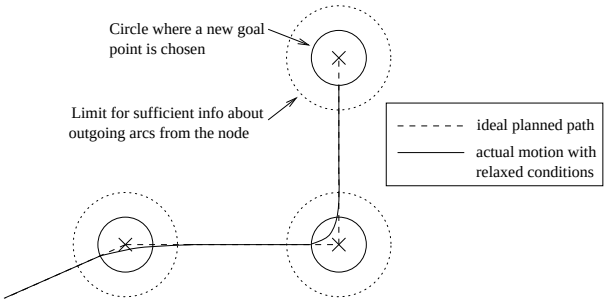

Figure 8: *Relaxation of the control problem.*

If the reference is smooth and long enough the method is guaranteed to work, but the switching of reference paths at the nodes could cause problems.

3.3 Example of Online Path Planning Combined with Control

The combined planning and control package was tested on the previous maze example.

In the simulation, a car-like robot model [1] is used:

$$\begin{cases} \dot{x} = v\ cos(\varphi + \psi) \\ \dot{y} = v\ sin(\varphi + \psi) \\ \dot{\psi} = \frac{v}{l} sin(\varphi) \end{cases} \quad (11)$$

where (x, y) is the mid point of the vehicle's front axle, φ is the steering angle and the other components have the same definitions as above. This model works for both forward and backward motion as well as switching continuously between them.

With this model we first need to find the corresponding steering control from the generic rotational velocity control:

$$\varphi = sin^{-1}(\frac{l}{v}(k\Delta\psi + \dot{\psi}_d)). \quad (12)$$

The simulation result is displayed in figure 9, where the steering angle is assumed to be saturated at $+/-\pi/5$. Figure a) is too small to show any details but as seen in b) the vehicle moves quite nicely along the planned path. It should be noted that the robot was relatively small (turning radius about 0.2 length units) and drove slowly, but the outcome of the test is undoubtedly positive.

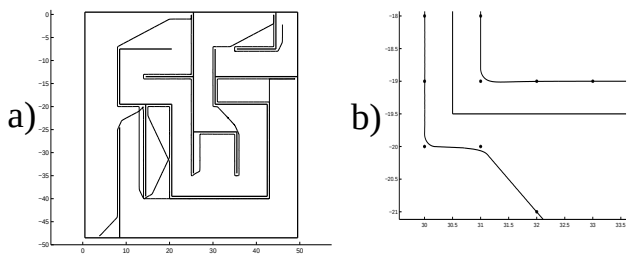

Figure 9: *a) The actual path through the maze. b) Zoomed detail of the maze example. An $*$ represents a node chosen by the planner.*

4 Conclusions

The paper presents an online path planner for unknown or partially unknown environments and shows that it can deal with really difficult problems, such as mazes. Furthermore, the applicability to navigation is demonstrated by combining the planner with path following control. Simulation results for the integrated online path planning and path following control module suggests that the method has a good chance of being feasible for real applications.

Acknowledgments

Special thanks to Dr. Stefan Feltenmark for his help with the code and his advice about network methods.

References

[1] J. Ackermann. *Robust Control.* Springer-Verlag, 1993.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: Theory, algorithms and applications.* Prentice Hall, 1993.

[3] M. Egerstedt, X. Hu, and A. Stotsky. Control of a car-like robot using a virtual vehicle approach. In *Proceedings of the 37th IEEE Conference on Decision and Control*, 1998. A revised version is to appear in IEEE-TAC.

[4] J-C Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[5] V.J. Lumelsky, S. Mukhopadhyay, and K. Sun. Dynamic path planning in sensor-based terrain acquisition. *IEEE Transactions on Robotics and Automation*, 6, 1990.

[6] B.J. Oommen, S.S. Iyengar, N.S.V. Rao, and R.L. Kashyap. Robot navigation in unknown terrain using learned visibility graphs. part i: The disjoint convex obstacle case. *IEEE Journal of Robotics and Automation*, RA-3 No.6 December, 1987.

[7] N.S.V. Rao. Algorithmic framework for learned robot navigation in unknown terrains. *IEEE Computer*, June, 1987.

[8] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings IEEE ICRA*, 1994.

[9] A. Stentz. The focussed $D^\star$ algorithm for real-time replanning. In *Proceedings of the Joint Conference on Artificial Intelligence*, 1995.

[10] A. Yahja, A. Stentz, S. Singh, and B.L. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environments. In *Proceedings ICRA*, 1998.

[11] J.S. Zelek. Complete real-time path planning during sensor-based discovery. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*, 1998.