Neural Maps for Mobile Robot Navigation

Michail G. Lagoudakis
Department of Computer Science
Duke University
Durham, NC 27708
mgl@cs.duke.edu

Anthony S. Maida
Center for Advanced Computer Studies
University of Southwestern Louisiana
Lafayette, LA 70504
maida@cacs.usl.edu

Abstract

Neural maps have been recently proposed [4] *as an alternative method for mobile robot path planning. However, these proposals are mostly theoretical and primarily concerned with biological plausibility. This paper addresses the applicability of neural maps to mobile robot navigation with focus on efficient implementations. It is suggested that neural maps offer a promising alternative compared to the traditional distance transform and harmonic function methods. Applications of neural maps are presented for both global and local navigation. Experimental results (both simulated and real-world on a Nomad 200 mobile robot) demonstrate the validity of the approach. Our work reveals that a key issue for success of the method is the organization of the map that needs to be optimized for the situation at hand.*

Introduction

Mobile robot navigation involves four basic subproblems: (1) Mapping, (2) Localization, (3) Path Planning, and (4) Motion Control. The complexity of addressing navigation at one level motivates a common practice of decomposing the problem hierarchically into local (sensory-based) and global (map-based) navigation. Local navigation focuses mostly on the last two subproblems, whereas global navigation focuses on the first three, path planning being their common ground.

Distance transform [5] and harmonic functions [2] are the most common methods used for path planning. Distance transform is a fast algorithm that unfortunately fails to produce smooth paths. Harmonic functions have the advantage of producing smooth paths, however at a prohibitive time cost. This paper contributes to the use of neural maps as a tool that 'hits' the middle ground and addresses its efficiency in real-time implementations. We claim that neural maps can be implemented in practice to produce smooth paths in short times. Moreover, we show how neural maps can be used for both global and local navigation, if adapted to the particular navigation circumstance. Simulation results are presented for the global case and results on a Nomad 200 mobile robot for the local case. Detailed description of our work and extensive discussion can be found in [6].

Neural Maps

A neural map is a *"localized neural representation of the signals in the outer world"* [1]. Signals from some space X are mapped through a mapping Ψ on a neural field F. F is a recurrent neural network, where the dynamics define lateral inhibitory or excitatory interactions among the units. *Amplification* (the mapping offers higher resolution to the 'critical' areas of X) and *neighborhood preservation* (geometric or functional proximity of signals in X is preserved as strong connectivity in F) are two important properties of neural maps. Additionally, the *self-organization* property is a kind of unsupervised learning that adapts the map dynamically in order to maintain amplification and neighborhood preservation. This combination of representational and computational mechanisms makes neural maps powerful tools in solving problems of practical interest.

The basic processing unit of a neural map is a non-linear 'integrate-and-fire' device. A unit i is characterized by the internal input vector V, the weight vector W_i, the external input θ_i, the net input u_i, the activation function $\Phi()$ and the output v_i. The net input u_i at time t is defined as

$$u_i(t) = W_i \times V(t) + \theta_i(t) = \sum_{j=1}^{n} w_{ij} v_j(t) + \theta_i(t)$$

The activation function $\Phi()$ can take several forms and it is usually a saturating nonlinear increasing function (e.g. sigmoid or hyperbolic tangent). The way the output of a unit i is updated over time is defined by the system dynamics. For discrete time, it is

$$v_i(t+1) = \Phi(u_i(t)) = \Phi\Big(\sum_{j=1}^{n} w_{ij} v_j(t) + \theta_i(t)\Big)$$

A large collection of such units (neurons) topologically ordered over X consists a neural map. The output (or state) of the map at any time t is the activation vector $V(t)$.

Neural Maps for Global Path Planning

It has been shown [4] that neural maps can be used effectively for path planning. In particular, the mapping resembles the n-dimensional configuration space C of a robot and the signal X is the current information about the robot's environment (free, obstacle and target configurations). The

0-7803-5529-6/99/$10.00 ©1999 IEEE

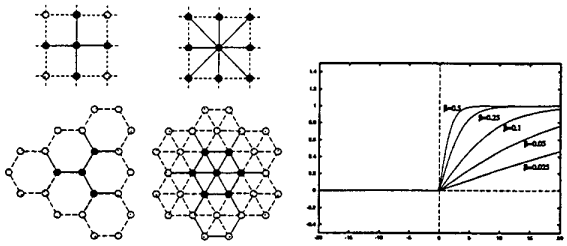

Figure 1: Connection topologies (left) and nonlinear activation function (right).

units of the network are distributed over C implementing a discrete topologically ordered representation of C. Thus, each unit i corresponds to a representative unique configuration c_i of the robot and each possible configuration in C is represented by the closest representative. The weight between units i and j reflects the cost of moving the robot from configuration c_i to configuration c_j; the higher the cost, the smaller the weight. Cost is defined in terms of some criterion to be optimized. The diffusive dynamics of the map are due to local lateral excitatory interactions and result in a stable activation landscape over C that provides a complete navigation map for the given target(s). A simple steepest ascent procedure on the equilibrium activation surface from *any* initial configuration will return an obstacle-free path to the (closest) target.

For holonomic mobile robots, C is the 2D plane. In general, the units of the map are distributed homogeneously over C. The topology and number of connections can varry. Common choices include rectangular or hexagonal topology with connections from (and to) unit i extended within a local neighborhood of units. Examples are given in Figure 1 (left). The common cost in mobile robot navigation is proportional to the Euclidean distance between configurations. Thus, the weight w_{ij} of the connection between units i and j is

$$w_{ij} = \begin{cases} 0 & \rho(c_i, c_j) = 0 \\ f(\rho(c_i, c_j)) & 0 < \rho(c_i, c_j) < r \\ 0 & r < \rho(c_i, c_j) \end{cases}$$

where $\rho(i, j)$ is the Euclidean distance between configurations c_i and c_j and $f()$ is a decreasing function. Common choices include $f(x) = 1/x$ (our choice), and $f(x) = e^{-\gamma x^2}$. Thus, all connections are excitatory, without self-coupling and local (within distance r).

The external input (e.g. sensory input) projected on unit i takes the form:

$$\theta_i(t) = \begin{cases} +\infty & i \text{ is a target configuration at time } t \\ -\infty & i \text{ is an obstructed config. at time } t \\ 0 & \text{otherwise} \end{cases}$$

The activation function (Figure 1, right)

$$\Phi_\beta(x) = \begin{cases} 0 & x \leq 0 \\ \tanh(\beta x) & x > 0 \end{cases}$$

defines the diffusive dynamics of the network. Notice that a target unit will be maximally activated ($v_i(t) = 1$), whereas an obstacle unit will be deactivated ($v_i(t) = 0$). All other units are not directly affected by the external input; their activation depends on the units in their neighborhood. The stability condition for the network is $\beta < 1/\lambda$, where β is the steepest slope of $\Phi()$ and $\lambda = \max\{|\lambda_i|\}$, where the λ_i's are the eigenvalues of the connectivity matrix. Intuitively, this condition guarantees that the activation of a unit i will not saturate even if all units in its neighborhood are maximally activated. Further, it is guaranteed that, for any combination of targets and obstacles, the map will converge to a unique stable state [4].

Suppose that all units are updated simultaneously in parallel during network evolution. In the absence of any target units, the network will eventually rest in the zero state, because the condition above imposes a strict gradual decrease of activation. When some target(s) is (are) specified activation spreads out of the target unit(s) in all possible directions, however without overpassing obstacle units. This wave propagation comes to a steady state, whereby the activation of a unit i is a measure of the shortest distance between configuration c_i and the (closest) target through an obstacle-free path; the higher the activation, the closer to the target. A steepest ascent procedure on the equilibrium activation surface from *any* initial position will return a collision-free path to the target (if such a path exists).

An example is given in Figure 2; the target is near the top-left corner, the initial position is near the top-right corner, and there are three wall-like obstacles. A 50×50 rectangular map ($r = 1.5, \beta = 0.1$) was used.

Efficient Implementation

It is important for real-time applications to minimize the convergence time of the neural map. A single unit is updated in constant time given the locality of connections. The model as presented assumes that all units are updated in parallel to ensure uniform activation propagation. On a conventional sequential computer the cost for simulating a single parallel update step is proportional to the total number of units in the map.

The knowledge of the robot's initial position can be used to terminate the evolution of the network before equilibrium [4]. Activation propagates from the target evenly in all directions. The direction of maximum gradient of the activation landscape at some unit will not change by the time the activation front 'hits' that unit. Therefore, network evolution can be terminated as soon as the activation front reaches the initial position. This reduces the number of parallel update steps to be proportional to the distance of the shortest obstacle-free path between initial position and target. Although this is a significant improvement, the bigger cost of a single parallel update step is not reduced.

The idea is to reject parallel updating altogether. This results from the observation that the equilibrium state is unique no matter how the network actually ends up there. We applied the Rosenberg-Pfaltz algorithm [7] (originally proposed for image distance transform) that performs sequential update rasters over the network. Successive rasters are applied along the 'diagonals' of the lattice structure. This way activation from a target unit is propagated along

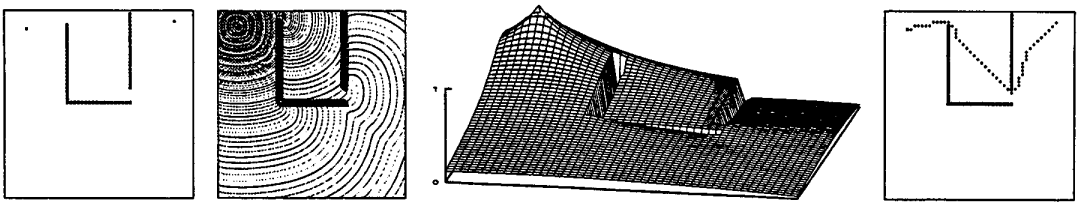

Figure 2: Target and obstacles (left), activation diffusion and equilibrium landscape (middle), and path (right).

the whole diagonal in one raster. By crossing successive rasters, activation spreads almost uniformly in all directions and covers rapidly the whole network. Convergence is achieved in significantly less time. An intuitive example here is the function of a paintbrush that distributes an initial concentration of paint on a surface through successive rasters in crossing directions. Notice that evolution can still be terminated early if the pattern of rasters tends to make propagation uniform and the test for termination is performed only after the pattern is completed. Such a pattern of four crossing rasters is shown in Figure 3, where the termination test can be applied every four rasters. The final path is close to the one derived by the equilibrium landscape.

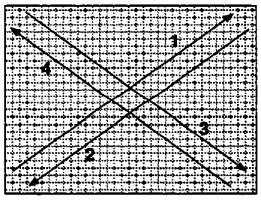

Figure 3: Pattern of sequential update rasters.

Results

By applying this technique, it was possible to generate paths in 2D neural maps of sizes 200 × 200, 500 × 500 or 1000 × 500 in less than 10 seconds given a simple arrangement of obstacle configurations. For maps of 50 × 50 and for any complicated arrangement of obstructed and target configurations the required time was about 1–2 seconds[1].

These time results compare favorably with the times reported in [2] (e.g. 188 seconds) for harmonic function-based path planning on 50 × 50 grids. Moreover, paths produced by the neural map are much smoother compared to the ones in [5] produced by distance transform on similar grids. The possibility of using interpolation methods to smooth out the activation landscape offers the opportunity for even better paths by using neural maps. Finally, the approximations proposed above are supported by the fact that in a dynamic environment global path planning takes place repeatedly; paths change continuously and the notion of optimality becomes hard to define.

[1]All runs were conducted on a SUN SPARCStation 4.

Neural Maps for Local Navigation

A major limitation of global navigation is the requirement for complete information about the environment. Local navigation uses only the available sensory information to guide the robot, however without any guarantee for completeness; even if a path exists, it may not be found. We have found the neural map method to be very efficient in this context, if adapted appropriately to account for the sensory and motor capabilities of the robot. We implemented a complete local navigation scheme that applies on a common class of mobile robots, namely, wheeled round nonholonomic mobile robots with range finding sensors (sonars and/or infrareds) distributed on their periphery.

The Polar Neural Map

The sensory system of the class of robots we consider has a circular range and the resolution of data points decays with the distance from the center of the robot. Thus, the robot has up to date information only for a portion of its configuration space and this information is not uniformly distributed. This motivated us to organize (manually) a neural map with a polar topology centered at the robot's center. The map 'moves' with the robot and at any time covers the area of the configuration space that is directly accessible through the sensory system. The distribution of units in the map resembles the distribution of the sensory data points.

Figure 4 (left) shows the polar map topology. The units have uniform angular and radial distributions. However, by changing the radial resolution, the map can capture larger or shorter areas of the real robot's surrounding up to a maximum radius determined by the sensor capabilities. Each unit in the map codes a possible position of the robot's center point in the real space. Therefore, obstructed configurations are determined as circular areas with radius equal to the robot's radius centered at the sensor's reading[2]. Figure 4 (left) shows also the *receptive field* of a unit which is the (sub)area of the signal space (sonar returns in our case) that is being mapped on that unit. As a result, any sonar return within that area (a circle with radius equal to the robot's radius) will characterize that position, and the associated unit, as obstructed.

The inner and the outer rings of the map cannot be occupied by obstacles because they have special roles. The

[2]We assume the simplest sensor model, where a sonar return corresponds to an obstacle lying along the main direction of the beam at the specified distance.

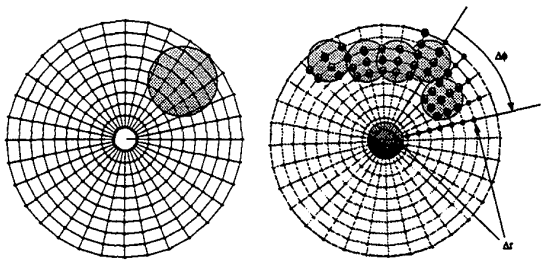

Figure 4: A simple 10x32 polar map (left) and an example of incremental path planning (right).

inner ring codes the center point of the robot and the robot's orientation. The outer ring is used for target direction specification, in cases where the target position falls outside the scope of the map. Having the sonar readings mapped and the target specified on the periphery (or inside the map), the dynamics of the network spread activation from the target unit(s) (in fact from outer rings to inner rings) until activation 'hits' the inner ring. The unit with the highest activation in the inner ring will determine the angular displacement $\Delta\phi$ from the current orientation and a steepest ascent procedure along this direction only will determine the radial displacement Δr. The pair ($\Delta\phi$, Δr) can be considered as a local subgoal and the cycle is repeated as the robot moves[3]. This way paths to the target are built incrementally step-by-step using only sensory information. Figure 4 illustrates the idea; five sonar returns from an L-shaped obstacle are mapped, the target direction is specified as an activated unit at the periphery, the path of maximum activation propagation is shown (notice that there is also another path from the left side which is longer) and finally the displacements ($\Delta\phi$, Δr) are displayed.

System Architecture

The complete architecture of our local navigation system is shown in Figure 5. Each component is described separately below. Briefly, the overall operation of the system is as follows: Some higher level component (a global path planner or a human supervisor) specifies a long-term goal, i.e. a target position given in polar coordinates with respect to the robot's egocentric frame. On the way to the target, the robot makes use of its odometry to measure its progress and update the position of the target with respect to its own egocentric frame of reference. The current velocities are used to predict the configuration of the robot at the end of the current control step and that position is taken as the current robot position. The current sonar readings are stored in the sensor short-term memory and all the contents of the memory are mapped on the polar neural map, using the predicted position as reference. The dynamics of the network spread activation from the target unit and the supervisor determines the short-term goal of the robot, as a displacement

[3]Notice that the nonholonomic constraints of the robot are postponed to the motion control level.

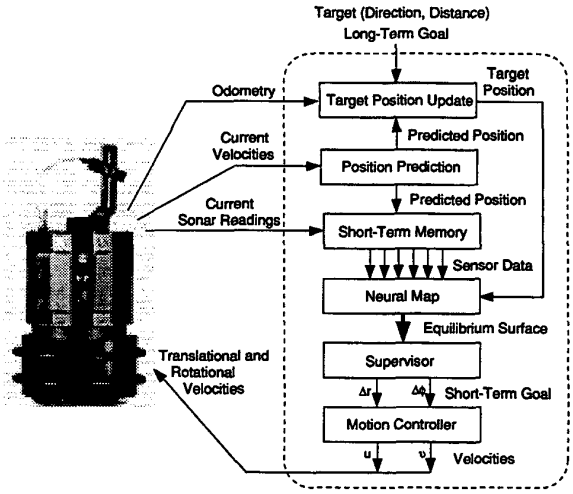

Figure 5: The complete architecture of the system.

Δr and $\Delta\phi$ with respect to the robot's current configuration. Δr and $\Delta\phi$ are passed to the motion controller which determines the appropriate control input and updates the robot's speeds. The whole cycle is repeated continuously until the target has been reached to a certain distance (currently 1 in). With the current implementation, the cycle time is approximately 0.33 sec.

Sonar Short-Term Memory

In order to cope with sonar noise, the system maintains a short-term memory of sensory information. In that sense, the most recent sonar returns (within the last 2-3 sec) are stored and reused in the next few steps before they are discarded. This short-term sensory integration provides a more accurate 'idea' of the robot's surrounding, without significantly reducing the ability to sense changes. Given the cycle time of our system, the 2-3 sec time window corresponds to a window of 6-10 sonar scans.

A sonar reading in the memory has been obtained at a configuration different than the one where it is reused, due to the robot's motion. Therefore, appropriate transformation is necessary. This is achieved by storing the configuration of the robot (from odometry) at the time each reading was obtained along with the readings themselves. Given this, all sensor readings, stored in the cyclical buffer that implements the memory, can be projected to any arbitrary configuration (in our case, the predicted configuration), as if they were obtained from there. For such a short time window, odometry can be safely assumed to be accurate and, thus, the transformation. Transformed readings that fall out of the map scope (the robot has moved away), or within the robot's physical dimensions (most likely erroneous ones), are discarded. Note that the system makes no assumptions about the shape and size of obstacles.

The effect of the short-term memory is demonstrated in

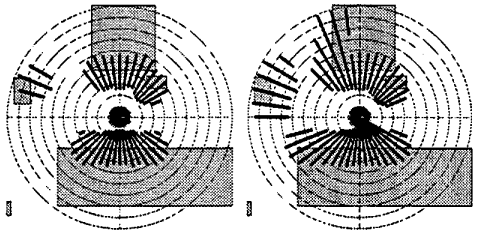

Figure 6: Representation on the polar map.

Figure 6. Assuming that the robot has arrived there from the top left corner, the instant sonar mapping (memory size=1) is shown on the left and the sonar mapping with a memory of size 10 on the right, both on a 100x48 polar map. The real obstacle positions are shown at the background shaded. Notice the richer representation in the second case and the fact that the robot is still aware of obstacles (like the left wall at the top) which are out of the sonar scope at the current position or non sensible because of the relative beam angle.

Configuration Prediction

It is an inherent characteristic of computer-based robot control that sensing and acting takes place at discrete times. Depending on the complexity of the software and the efficiency of the hardware, the time Δt between consecutive control steps might be large enough to lead to significant errors especially for high speed motion. The problem is that action at the end of the current control step corresponds to the environment as it was perceived at the end of the previous step.

The solution we propose is as follows. The system estimates (using its real-time clock) the time Δt between issuing control commands. Using the kinematic model of the robot (the unicycle model, in our case), the current velocities (from internal state), the current configuration (from odometry) and the time Δt, the configuration of the robot by the end of the control cycle (when the control command is issued) can be predicted. Using this configuration as reference the system projects all its data into the near future and the control command is determined with respect to that situation. Consequently, the resulting control command is not out of date by the time it is issued.

Given that Δt may not be constant but variable, an estimation is calculated continuously based on averages over several recent control steps. The advantage of this technique is that the control algorithm is being adapted continuously to the speed of the platform it is running on. Nevertheless, the faster the platform, the better the control.

Motion Control

Motion control is the problem of determining the appropriate control input (translational and rotational velocity in our case) that will drive the robot to the (sub)goals specified by the path planner. Our controller that takes into account the dynamic constraints of the robot is only outlined here.

Given the subgoal $(\Delta\phi, \Delta r)$ at the current step, we want to derive the control input (u, v) (translational and rotational velocities) for that step. The algorithm runs as follows:

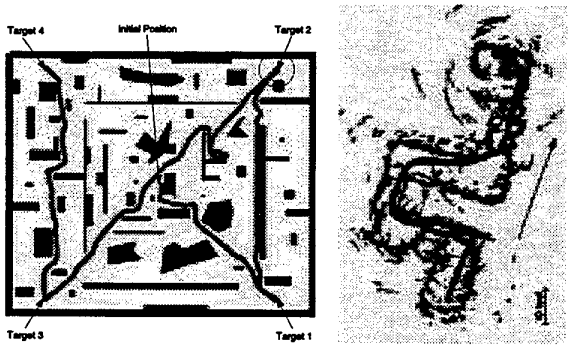

Figure 7: Sample runs of the robot in simulated (left) and real (right) worlds.

1. Determine the current Dynamic Window (DW) [3], i.e. the subspace of the velocity space directly accessible in one step, according to the maximum accelerations of the robot.
2. Apply the objective function on each pair of a discretization of DW. For each pair (u, v), the final configuration of the robot is estimated, firstly by invoking the trajectory equations of the unicycle kinematic model (for time Δt, the control step duration), and then by calculating the minimum braking distance and angle the robot will travel before it comes to a complete stop (assuming maximum deceleration). The objective function combines the distance between the desired configuration vector and the (estimated) final configuration vector with the weighted density of obstacles (given by the polar map) along this trajectory[4], to derive a measure of appropriateness of the pair (u, v).
3. Select the pair (u, v) with the minimum value as the next control input.

In effect, the robot is 'chasing' the subgoal returned by the polar map aiming to stop there. Since the subgoal moves incrementally, the resulting motion is smooth and continuous until the robot reaches the target, where it stops.

Results

Our scheme was tested on a Nomad 200 mobile robot of the Robotics and Automation Lab at USL. The Nomad has a ring of 16 sonars and can translate with a maximum speed of 24 in/sec and rotate with a maximum speed of 45 deg/sec.

Figure 7 (left) shows a sample (simulated) run of the robot with four target positions. The thickness of the path trace is equal to the robot's diameter and the circle at the top right indicates the sensory/map scope used (100 in). A polar map of size 100x32 has been used in this and all subsequent runs.

Figure 7 (right) is a real-world run in a typical lab/office environment (USL Robotics and Automation Laboratory).

[4]Note that this is necessary due to the nonholonomic constraints in order to avoid collisions.

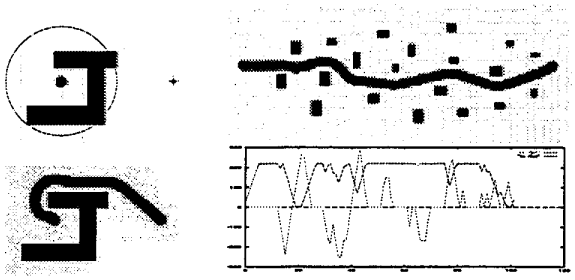

Figure 8: Avoiding a U-shaped obstacle (left) and navigation in a cluttered environment (right).

The robot started at the bottom and was given a distant target in the direction specified by the arrow.

Figure 8 (left) displays a typical U-shaped obstacle situation and the path followed by the robot. The circle indicates the sensory scope of the robot. Since our system can support only local navigation, such traps can be avoided only if the whole situation falls within the sensory range (and thus, within the polar map) of the robot. In the opposite case (large rooms, long walls, etc.) the robot will be trapped, since it is memoryless in the long term.

Figure 8 (right) shows navigation in a cluttered environment and the corresponding control input for this run. The dark line indicates the translational speed (0.1 in/sec) and the light one the rotational speed (0.1 deg/sec). The horizontal axis indicates control steps.

Figure 9 shows a possible extension to global navigation. A long trip in a simulated office-like environment is completed by passing (currently manually) only 13 subgoal positions to the local navigation system. Notice that these positions are certain landmarks (doors, junctions, etc.). Moreover, if the next subgoal in the sequence is given shortly before the previous one has been reached, the result will be a continuous and smooth path.

The system works well enough with moving obstacles as long as they move with a speed comparable to the robot's speed. In particular, the robot had no trouble avoiding walking people in the lab.

Conclusion and Future Work

We have presented applications of neural maps in mobile robot navigation. It was suggested that neural maps offer an intermediate solution between distance transform and harmonic functions. We also proposed methods for efficient implementations and demonstrated the validity of the approach on a Nomad 200 mobile robot.

In the future, we would like to investigate the role of the weight values in the map for path construction. We believe that weights can be used to code for other factors, like difficulty of navigating in different parts of the plane, preferable pathways, asymmetric costs like inclines, etc. Also, for faster path construction we are interested in heuristics for updating only an adequate portion of the map. Such ideas have already appeared [8] in the context of A* search. For

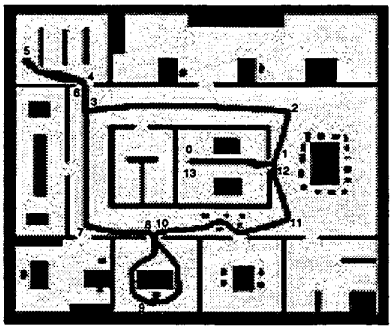

Figure 9: Extension to global navigation.

the polar map, we plan to (re)organize it into a polar and logarithmic topology that more closely follows the sonar data distribution. Finally, we would like to make use of the self-organization property to make the map self-adaptive to the robot's sensory system. The ultimate goal is an integrated system for full robot navigation based on neural maps.

Acknowledgments

The first author would like to thank the Lilian-Boudouri Foundation in Greece for financial support. Special thanks to Prof. Kimon P. Valavanis, director of the USL Robotics and Automation Lab, for use of the facilities.

References

1. Amari, S. 1989. Dynamical Stability of Formation of Cortical Maps. In *Dynamic Interaction Neural Networks: Models and Data*, eds. M. Arbib and S. Amari, Springer-Verlag, pp. 15–34.
2. Connoly, C; Burns, J; and Weiss, R. 1990. Path Planning with Laplace's Equation. *IEEE International Conference on Robotics and Automation 1990*, pp. 2102–2106.
3. Fox, D.; Burgard, W.; and Thrun, S. 1997. The Dynamic Window Approach to Collision Avoidance. *IEEE Journal of Robotics and Automation*, 4, 1, pp. 23–33.
4. Glasius, R.; Komoda, A.; and Gielen, S. 1995. Neural Network Dynamics for Path Planning and Obstacle Avoidance. *Neural Networks* 8, 1, pp. 125–133.
5. Jarvis, R. 1993. Distance Transform Based Path Planning for Robot Navigation. In *Recent Trends in Mobile Robots*, ed. Y. Zheng, World Scientific Pub, pp. 3–31.
6. Lagoudakis, M. 1998. *Mobile Robot Local Navigation with a Polar Neural Map*. M.Sc. thesis, University of Southwestern Louisiana.
7. Rosenfeld, A., and Pfaltz, J. 1966. Sequential Operations in Digital Image Processing. *Journal of the Association for Computing Machinery*, 13, 4, pp. 471–494.
8. Trovato, K. 1996. *A* Planning in Discrete Configuration Spaces of Autonomous Systems*. Doctoral Dissertation, University of Amsterdam, Netherlands.