Simple Robots in a Complex World: Collaborative Exploration Behavior using Genetic Programming

Keith Ito
Department of Computer Science
Stanford University
kito@cs.stanford.edu

Abstract

Mapping unknown environments is a key problem in mobile robotics. This paper applies genetic programming to evolve controllers for robots in a collaborative mapping scenario. A program that outperforms human-designed controllers using the same data inputs was evolved, suggesting that genetic programming may be a feasible approach for designing mapping behaviors.

1 Introduction

Mapping unknown environments is one of the key challenges in mobile robotics. The accurate construction of maps is a prerequisite for navigation and path planning in mobile robot systems such as Helpmate [2]. Mapping can also be an application in itself, for example in the creation of VRML "fly-throughs" as described by Thrun [8].

An important part of efficiently mapping an environment is selecting an exploration strategy that allows a robot to sample the entire environment with distance sensors in a minimum amount of time. Collaborative mapping uses multiple autonomous robots to cover a larger area of the map per unit time. This can result in faster map generation, but also introduces the challenge of minimizing the overlap between regions explored by individual robots.

This paper applies genetic programming (GP) to the mapping problem. Specifically, it uses GP to evolve a control module for a small team of autonomous robots such that the maximum area of an unknown environment is mapped in a fixed amount of time. It demonstrates the feasibility of using GP to evolve mapping behaviors, and also produces a control module that exhibits complex and seemingly intelligent global behavior despite making only simple local decisions.

1.1 Related Work

1.1.1 Robotic Mapping

The mapping problem has been addressed thoroughly with human-designed algorithms, both in the single-robot case and in multi-robot groups. Much of this work has focused on localization and data noise issues, such as dealing with the accumulation of odometry errors which can lead to increasing uncertainty as time progresses. Numerous algorithms have been devloped for maintaining accurate global localization [6], or using probabilistic techniques to make such information unnecessary for generating a global map [9]. This paper assumes that such techniques (or alternately a system such as GPS) is used to eliminate the accumulation of error so that the robot's global position at a given instant in time can be modeled as a simple Gaussian distribution about a point.

Methods for controlling the behavior of mapping robots range from simple, local algorithms such as random walk [1] or wall-following [6] to centralized, frontier-based algorithms that attempt to plan globally optimal paths for each robot. Simmons et. al. [7] present a method of maximizing the expected information gain that a path of exploration will yield by solving an optimization problem at a centralized controller. This type of approach is capable of very rapidly navigating robots to the unexplored frontier to gather new information.

Objective	Evolve a control program for exploration and mapping by a small group of robots
Terminal Set	The distance to the nearest wall in 5 directions (`S0...S4`), evidence grid density estimates at each sensor hit (`D0...D4`), actions: Move Forward (`MF`), Turn Right (`TR`), Turn Left (`TL`).
Function Set	If-Less-Than-Or-Equal(`IFLTE`), and the connective (`PROG2`)
Fitness Cases	One to four (depending on the run), a combination of the "room" and "maze" map types (see section 2.4).
Raw Fitness	The fraction of the ground truth map that was successfully mapped by the robot's sensors within a small error radius.
Standardized Fitness	One minus raw fitness.
Parameters	M = 100 to 250 (depending on run), G = 101.
Success Predicate	A robot maps 90% of all sample points before time expires (due to the map representation, some points on the map are unreachable, and 90% represents close to a complete solution).

Table 1: Genetic programming tableau for the robotic mapping problem.

A disadvantage is that it requires the robots to be synchronized with the controller and to be aware of their relative locations. In addition, it requires that the robots have a significant processing resources on-board.

1.1.2 Genetic Programming

Genetic programming has been applied to a number of robotics problems, including the learning of simple behaviors such as obstacle avoidance [5] and wall following. Koza [3] presents an application of GP to teach a robot to navigate around the periphery of a square room with small protrusions in the walls. Lazarus and Hu [4] extend this work, demonstrating that additional protrusions in the walls are tolerable. However, none of these GP approaches have been applied to mapping, which presents a number of additional problems.

2 Methods

This paper takes the representation used by Koza [3], and modifies it to handle the additional challenges presented by the mapping problem, such as navigating through oddly-shaped, previously unseen rooms, moving from room to room, and avoiding areas that have already been mapped. It does so by adding additional terminals that allow the algorithm to access an evidence grid similar to that described by Yamauchi [10]. and redefining the fitness evaluation procedure to measure map quality.

The specifics of the genetic programming approach used in this paper are summarized in tableau form in Table 2, and described in greater detail below.

2.1 Terminals

Terminals are used to represent both sensor readings and actions that can be taken by each robot.

The robot is provided with 10 sensor readings: the distances to the nearest objects in five directions, as well as estimates of the density of the evidence grid at each of those objects. The distance terminals, labeled `S0, S1, S2, S3,` and `S4` return the distance to the nearest obstacle in the left, front-left, front, front-right, and right directions, respectively, and could be used by the robot in obstacle avoidance, boundary tracking, or any number of functions. The density terminals, labeled `D0, D1, D2, D3,` and `D4` return estimates of the density of mapped points at each of the locations corresponding to the five distance readings. These values might be used by the robots to seek areas that have not yet been explored or to find return paths.

The robot can take three actions, also represented by terminals: it can move forward (the `MF` terminal), turn left (`TL`), and turn right (`TR`). Each action terminal produces the indicated side effect when evaluated and returns 0.

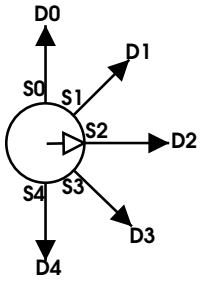

Figure 1: The simulated robot has five distance sensors, labeled `S0...S5` and the five evidence density pseudo-sensors, labeled `D0...D5` spaced at 45 degree angles along the front of the robot.

2.2 Functions

The `(IFLTE arg0 arg1 eval0 eval1)` function compares `arg0` and `arg1`, and if the former is less than or equal to the latter, it evaluates and returns `eval0`. If not it evaluates and returns `eval1`. This allows for conditional branching and assignment.

The `(PROG2 eval0 eval1)` function evaluates both its arguments and returns the sum of their return values. This can be used to chain together multiple execution trees and also functions as an addition operator.

These are the same functions that were used by Koza in the wall following problem. The justifications Koza provides - that it is desirable to simplify the function set in genetic programming, and that all numerical comparisons can be reduced to $\leq$ with reordered arguments also apply in the mapping case.

One modification is that the return value of the `PROG2` function is defined as the sum of its arguments rather than the value of the latter. This allows multiple sensor readings to be easily added to create more complicated comparisons.

2.3 Robot simulator

A discrete-time simulator was constructed for the purpose of fitness evaluation. In the simulator, each robot is modeled as an oriented point. Thus, a robot is parametrized by three continuous values, an x-location, a y-location, and a direction. While this representation is somewhat unrealistic, any solution found here can be mapped into a space with finite-dimensional robots by treating the robot location as the center of the robot and surrounding the walls with shells at a distance equal to the robot's radius.

Each robot is capable of executing one action (moving forward two units, turning left 30 degrees, or turning right 30 degrees) each time step.

To generate sensor readings for each robot, the simulator intersects rays emanating from the robot's location in five directions (`S0` (left): $90°$, `S1` (front-left): $-45°$, `S2` (front): $0°$, `S3` (front-right): $45°$, and `S4` (right): $-90°$) with the geometry of the map to compute the exact distance. A diagram of these sensors and their orientations is shown in Figure 1.

Sensor noise is then introduced to these distance readings by adding samples from a Gaussian distribution with mean 0 and a standard deviation proportional to the magnitude of the reading. In addition, localization noise is modeled by perturbing the recorded point by a sample from a 2-D Gaussian distribution.

The simulator also generates evidence density estimates for use by the robot control program. This is implemented by dividing the map into a number of coarse uniform subgrids (about 1000) and counting the number of sampled map points that fall into each subgrid. To generate a density reading, the simulator maps the endpoint of the ray to a grid cell and returns the value for that cell.

Finally, the simulator evaluates the GP individual's program tree once for each robot in the simulation and applies the actions to update the robot's location and orientation.

2.4 Map generation

To avoid a solution that overfits a specific training example by learning behaviors that are only beneficial on that particular map, each generation was run on a set of different, randomly generated maps. These maps

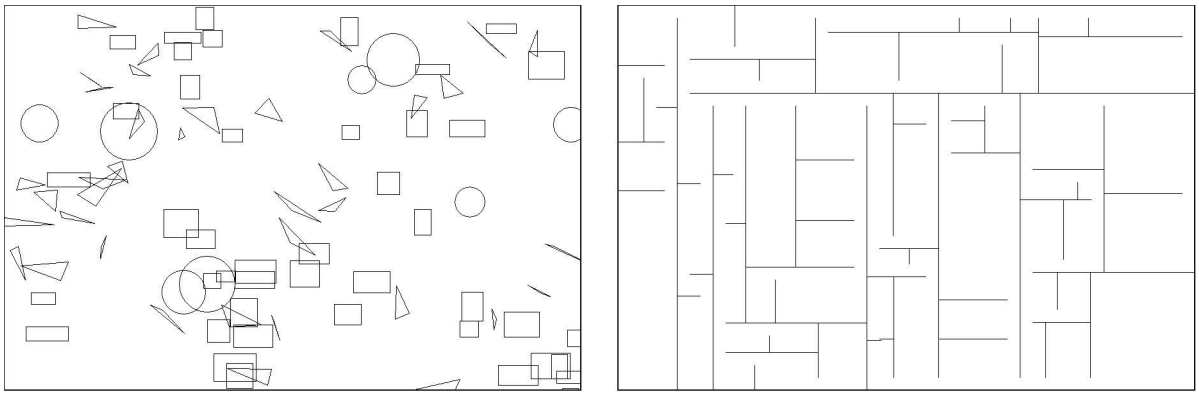

Figure 2: Examples of a randomly generated "room" map (left) with parameter *obstacles* = 90, and a "maze" map (right) with parameter *walls* = 160.

were created by the following two algorithms:

2.4.1 The "room" algorithm

This algorithm simulates a cluttered room or debris field by placing a given number of randomly located obstacles into the map area. Each obstacle can be either a circle, triangle, or rectangle, and the size of each of these varies. No care is taken to ensure that obstacles do not overlap, which renders some edges of some obstacles unreachable, and as a result the maximum attainable fitness in this type of map is less than 1. The left panel in Figure 2 shows an example room.

2.4.2 The "maze" algorithm

This algorithm simulates a building or city by generating a maze-like structure with hallways and rooms. This is accomplished by starting with a box and repeatedly randomly selecting a point on one of the walls as a split node. A new wall is generated from that point to the first perpendicular wall encountered, leaving space at the end to allow the robot to pass through. The algorithm ensures that the entire map is reachable from any starting point. The right panel in Figure 2 shows an example maze.

To demonstrate that a GP approach could work in complex environments, maps were chosen to be large, with a width of 1000 units and a height of 800 units, and complex, with similar control parameters to those in Figure 2.

2.5 Fitness evaluation

Fitness is defined as the fraction of the map that is successfully mapped by an individual. This fitness measure is implemented by taking a large uniform (over length) random sample of points from the geometry of the map and counting the number of these samples that were mapped by the robots within a small error radius ($\tilde{2}$ units). The greater the number of samples hit, the more complete the map is and the more adept the individual is at mapping in this environment.

2.6 Parameters

Parameters were chosen to balance the increased diversity gained by having a large population with practical time constraints imposed by the problem. Since each time step requires 15 rays to be traced through the environment, and since the implementation was written in Java, generation times could be lengthy. As a result the population size was set to be 150 individuals on most runs. Although this is less than the 500 individual size recommended by Koza, it was necessary to have runs complete in a reasonable amount of time.

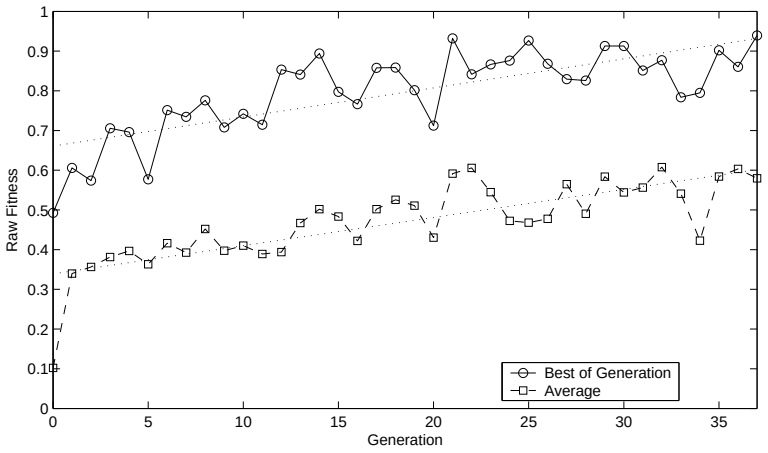

Figure 3: Performance of the best and average individuals on one run. Due to the randomization of fitness cases, the curves are noisy, yet show a general increase in successive generations. Least squares fits of each curve are shown in dotted lines.

Individuals were given 10,000 time units to complete their traversal of the map. This was chosen to be roughly triple the amount of time that an optimal solution would take visit the entire map in the "maze" case.

The crossover percentage was set to 90%, and the mutation rate was set to 0%, although single node crossover, which has a similar effect to mutation, was allowed.

3 Results and Discussion

Approximately ten runs of the genetic programming instance described in the previous section were run under slight variation in parameters such as population size, simulator sensor noise, and simulation timeout. Runs lasted between 10 and 25 hours on a dual UltraSPARC III / 900 MHz Solaris workstation with 2GB RAM, depending on the run parameters. The workstation was shared, so the running times may not reflect full processor utilization.

The results of one such run are shown in Figure 3. This run used a a population size of 100, a time limit of 15,000 time steps, and two randomly generated fitness cases per generation, one of each variety. There is a significant amount of noise in the results due to the randomization in fitness cases; by chance some are easier to map than others. However, the least-squares fits for the best-of-generation and average curves show that fitness generally improves as more generations are run. The best individual in generation 37 came close enough to completely producing correct maps of both environments to trigger the termination condition.

Results from several other runs were similar, although they required a greater number of generations to complete. Many runs, however, prematurely converged to suboptimal solutions. This may be due to the fact that a small population size was used and not enough genetic diversity was present to solve the problem.

3.1 Best-of-run individual analysis

Some individuals generated by the GP process exhibited surprisingly complex behavior despite a simple encoding and low computational cost. The best individual from the the run graphed in Figure 3 is one such example. It is shown below.

```
(IFLTE
  (IFLTE D2 S3 D1 D2)
  (IFLTE D3 (IFLTE D4 D3 S4 S1) MF D3)
  (IFLTE D4 D4
    (PROG2
      (IFLTE D3 D2
```

```
        (PROG2 TL D0)
        (IFLTE D1 D2 S3 (PROG2 D4 (IFLTE S4 MF S3 D2))))
      MF)
    (IFLTE
      (IFLTE D3 D2 S3 (PROG2 S3 S0)) D4
      (PROG2 TL MF)
      (IFLTE D3 D2 S3 S0))))
(PROG2 D0 TR))
```

When clauses with no side effects that are unreachable or do not return a value are removed, this expression simplifies to the following.

```
(IFLTE
  (IFLTE D2 S3 D1 D2)
  (IFLTE D3 (IFLTE D4 D3 S4 S1) MF D3)
  (PROG2
    (IFLTE D3 D2 (PROG2 TL D0) (IFLTE D1 D2 S3 MF))
    MF)
  TR)
```

The behavior produced by this control program can be roughly paraphrased as:

- If you are very far from the nearest wall to your right (and the density to the right is not greater it is to the front), then *move forward* and *turn right*.
- Otherwise, if the density is higher to the left than it is to the right, then *turn right*.
- Otherwise, if the density is higher to the front than it is to the right, then *turn left* and *move forward*.
- Otherwise, if the density is higher to the left than it is to the front, then *move forward* twice.
- Otherwise, *move forward* once.

These control rules lead to a cumulative behavior in which the robot first tries to follow walls (keeping the wall on its right side), and, once it has met this goal, attempts to seek out the regions of the map that have been least explored, with a preference toward regions to its right.

This is an elegant solution that relies only on local decisions to generate a seemingly intelligent aggregate global behavior. In its simplified form, the solution requires only between 6 and 14 operations (comparisons, adds, copies, etc.) per iteration.

An important observation is that the control program evolved what is essentially a three-argument minimum function. By performing pairwise comparisons between between `D1`, `D2` and `D3`, the control program correctly selects the minimum of the three in 5 of 6 potential cases (when the ordering is `D1` $>$ `D3` $>$ `D2`, the first comparison will short-circuit and `D3` will be erroneously selected as the minimum).

Also interesting is the fact that despite efforts to randomize the training cases to avoid an overfit solution, the GP was still able to identify and exploit a common characteristic among them: the lack of large open spaces in the maps. Suppose the robot is placed in a large open space, i.e. it is far from walls on all sides. Then, following the first point in the text description of the control program, the robot will move forward and turn right since it is far from the nearest wall on its right. After this maneuver, it will *still* be far from the nearest wall on its right, so it will repeat the maneuver. This will continue, and the robot will spin in circles.

If a large room without walls is constructed, and a single robot is placed in the center, this does in fact occur for several thousand time steps until, due to interactions between density to the right (`D4`) and density to the front-right (`D3`), it is able to break out of the loop and drift to the periphery where it commences a wall following behavior. Despite its ability to break out of this loop, a large amount of time was wasted in doing so. If one of the fitness cases was defined to have a large open space, the behavior would probably have evolved to handle such a situation better.

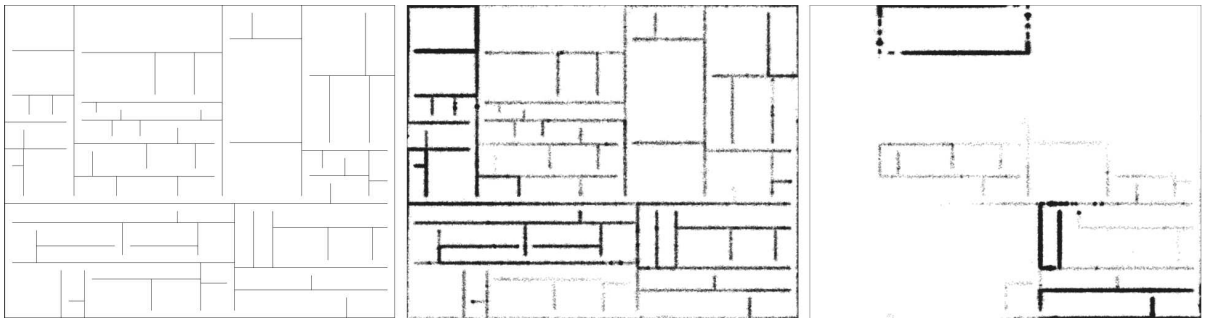

Figure 4: Performance on a randomly generated map. The ground truth map (left) is shown along with a map produced by the best-of-run GP individual (center), and a map produced by a robot with wall-following behavior (right). Both the GP and wall-follower maps were the result of three robots running for 10,000 time units.

3.2 Quantitative comparison with human-designed behavior

A human programmer assigned the task of writing an exploratory behavior for a robot with the same sensors and actions as described in Section 2 might consider a number of strategies. One justifiable choice would be wall following since this seems to work well, at least in single-room maps.

To compare such a human-designed solution with the GP-evolved solution, we implemented a wall following control program for the simulated robot. These two programs were then run on a randomly-generated environment that was previously unseen by the GP. Each was given three robots, randomly placed in the environment, and 10,000 time units to generate as complete a map of the environment as possible. The starting locations of the robots and all other parameters other than the control program were held constant between the two runs. Results are shown in Figure 4. For this particular example, the evolved individual had successfully mapped 96.3% of the environment when time expired, while the wall follower had only mapped 38.6%. The figure shows that the GP-evolved controller's map is nearly complete, missing only a one small room on the right side. It is also fairly uniformly dense, with samples that are not overly concentrated in any single room. The wall follower controller's map, on the other hand, is incomplete, leaving out large sections of the environment, and samples are overly concentrated in several rooms.

Examining the time course of the run for each control program, as shown in Figure 5, sheds light on why the GP-evolved program outperforms the wall follower. While both programs are able to quickly map the areas around the starting locations of the robots, the wall follower quickly loses the ability to find new areas to map, and its fitness levels off. In the "Maze" map, the wall follower tends to get stuck in rooms with only a single entrance and a narrow doorway, and in the "Room" map, it never ventures into disconnected areas, missing certain edges in the environment. Since the GP-evolved program is constantly looking for unmapped areas, it does not get stuck in these situations. This gives it a significant advantage over the wall follower, particularly as time progresses.

4 Future Work

Although multiple robots are used for mapping, all of them in a given run are controlled by the same algorithm, the one encoded in the individual whose fitness is being evaluated. Thus, if placed in a similar location with similar inputs, all robots in the run would be expected exhibit the same behavior. Significant advantages might arise from the ability to run different control programs on each robot.

We implemented two variants of the genetic algorithm to test this hypothesis. The first variant altered the definition of an individual to include three control trees, one for each of the three robots in the simulation. This allowed each robot to run a different program. All individuals were still considered part of a single population, and crossover could occur between any two branches in a mating pair.

The second variant employed cooperative coevolution to generate separate behaviors for each of the three robots. The individuals were separated into three distinct pools; individuals in a pool could only mate with other individuals in the same pool. Fitness for members of a given pool was determined by running the

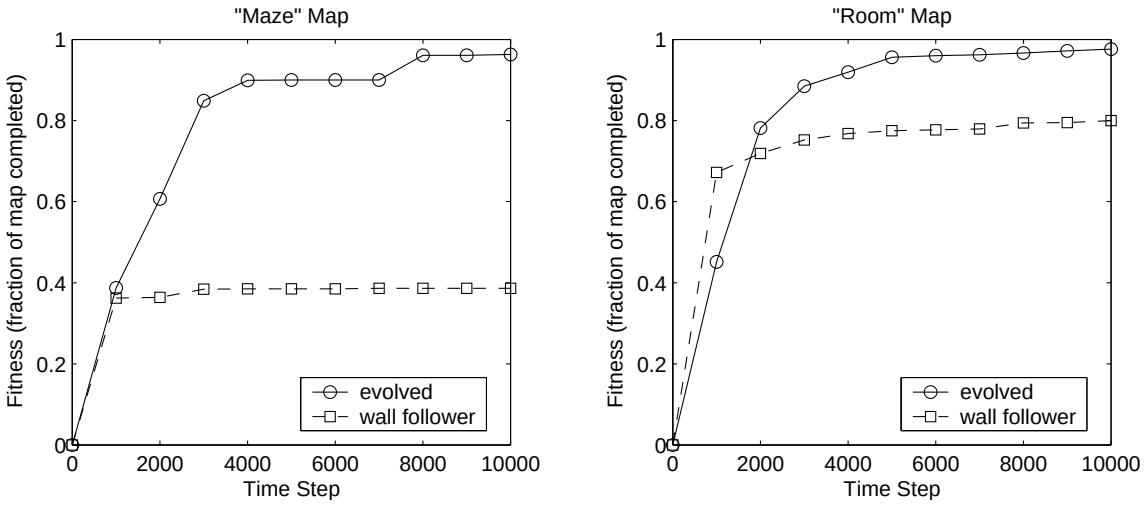

Figure 5: Comparison with human-designed behavior. This graph compares the fitness at various time points on two runs of the GP-evolved controller and the human-coded wall follower controller. The left graph shows results on a "maze" type map, and the right shows a "room" type map. In both cases, the wall follower and the GP individual start off mapping the environment at roughly the same rate. However, the GP individual is able to identify areas that still need to be mapped, improving its fitness as time progresses, while the wall follower cannot.

individual with the best-of-generation individuals from the other two pools and calculating fitness as before.

Initial experiments using these variants with small population sizes and short simulation timeouts did not show a significant improvement in fitness for either method.

Due to a lack of time and computational resources, these experiments could not be repeated with longer runs or larger populations. Performing these experiments and analyzing the results could shed additional light on more efficient ways of using genetic programming in mapping applications.

5 Conclusions

This paper presents the use of genetic programming to generate an efficient behavior for the exploration of large, complex environments for the purposes of mapping. One control program produced by a genetic programming run exhibited a mixture of wall-following, frontier-based exploration, and random diffusion behaviors. It was successful in efficiently mapping its environment and significantly outperformed simple human-coded behaviors such as wall following and random walk. These results demonstrate that the genetic programming approach to evolving behavior controls may be an attractive solution to the mapping problem, particularly when centralized control of the robots is not an option.

References

[1] Dellaert, F., F. Alegre and E. Martinson. 2003. Intrinsic Localization and Mapping with 2 Applications: Diffusion Mapping and Marco Polo Localization. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

[2] King, S. and C. Weiman. 1990. Helpmate autonomous mobile robot navigation system. In *Proceedings of the SPIE Conference on Mobile Robots*. Boston, MA. Pages 190-198.

[3] Koza, J. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge: The MIT Press.

[4] Lazarus, C. and H. Hu. 2001. Using Genetic Programming to Evolve Robot Behaviours. In *Proceedings of the 3rd British Conference on Autonomous Mobile Robotics and Autonomous Systems*. Manchester.

[5] Nordin, P. and W. Banzhaf. 1995. Genetic Programming Controlling a Miniature Robot. In *Working Notes for the AAAI Symposium on Genetic Programming* Cambridge, MA: MIT Press. Pages 61-67.

[6] Rekleitis, I., G. Dudek, and E. Milios. 2001. Multi-Robot Collaboration for Robust Exploration *Annals of Mathematics and Artificial Intelligence*. 31:1, pages 7-40.

[7] Simmons, R. et. al. 2000. Coordination for Multi-Robot Exploration and Mapping. In *Proceedings of the 12th Innovative Applications of AI Conference*.

[8] Thrun, S. 2001. An Online Mapping Algorithm for Teams of Mobile Robots. Algorithm for Teams of Mobile Robots *Journal of Robotics Research*. 20(5):335-363.

[9] Thrun, S., W. Burgard and D. Fox. A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots. *Machine Learning and Autonomous Robots*. 31(5): 1-25.

[10] Yamauchi, B. 1998. Frontier-Based Exploration Using Multiple Robots In *Proceedings of the Second International Conference on Autonomous Agents* Monterrey, CA. pages 146-151.