IEEE/RSJ International Conference on Intelligent Robots and Systems, 1992.

The Learning of Reactive Control Parameters Through Genetic Algorithms

Michael Pearce, Ronald Arkin, and Ashwin Ram
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332 U.S.A.

***Abstract*—This paper explores the application of genetic algorithms to the learning of local robot navigation behaviors for reactive control systems. Our approach is to train a reactive control system in various types of environments, thus creating a set of "ecological niches" that can be used in similar environments. The use of genetic algorithms as an unsupervised learning method for a reactive control architecture greatly reduces the effort required to configure a navigation system. Findings from computer simulations of robot navigation through various types of environments are presented.**

I. Introduction

A common robot task is to navigate through an environment to a goal position, without hitting any obstacles that may be present. Navigation through a cluttered environment is an extremely complex and underconstrained task. Apart from the computational constraints placed on the design of a navigation system, it is desirable that the system be robust enough to navigate through a large number of possible environment configurations.

Traditional robotics research has focussed on symbolic representations and world modeling to solve navigation problems [1, 8]. A large part of the work that these systems perform concentrates on the mapping of incoming sensor data to a high-level symbolic representation of the environment. Such systems combine data from different types of sensors, process this data to keep their internal world models up to date, and perform path planning from this world model. While these systems perform well in constrained environments, their performance is less robust in dynamic, real world environments. Symbolic robotic navigation systems rarely meet real-time constraints in tasks that seem trivial to humans, and when they do it is under highly constrained and closely supervised conditions [7].

An alternative approach to robot navigation is reactive control, which attempts to transform sensor data into information that directly affects the behavior of the robot. The behavior of these robot systems emerges from the careful organization of much simpler behaviors. These behaviors are designed so as not to require a world model, and are computationally less demanding than those of a symbol processing robot navigator. These architectures are able to act in real-time, and are more robust in dynamic environments than their symbolic architectural counterparts.

This paper describes the application of genetic algorithms to the problem of optimizing robot navigation control parameters in a reactive control system. Our approach is to train a reactive control system in various types of environments, thus creating a set of "ecological niches" that can be used in similar environments that were not presented in the learning phase. The use of genetic algorithms as an unsupervised learning method for a reactive control architecture greatly reduces the effort required to configure a navigation system. Since the genetic algorithms are run as simulations on a computer and do not require that the learning occur on the actual robotic system, they greatly decrease the amount of time required to present the system with a sufficient number of learning trials.

II. Related Work

A. *Reactive Control for Robot Navigation*

The reactive control approach to robotic navigation grew out of a dissatisfaction with traditional robotics architectures. Reactive control draws from the Behaviorist school of psychology, in that there are no explicit symbolic representations of the external world. The subsumption architecture [5], which typifies the purely reactive control model, is composed of simple behaviors (like wandering, obstacle avoidance, and goal following) that combine to produce "emergent behaviors" that were not explicitly designed to be exhibited by the system. The simple behaviors of a reactive control system acquire the information about the environment directly from the sensors, instead of getting this information indirectly through an intervening world model. These behaviors are closely tied to the effectors that carry out the behavior of the robot [12, 11].

These non-representational systems avoid many of the pitfalls experienced by the traditional symbolic based and world-model driven systems, but they do have their own problems. The subsumption architecture does not allow for the explicit representation of high-level goals, so it is difficult to reconfigure the systems to perform different tasks or reason about unperceived objects. Also, as behaviors are added to the system and it becomes more complex, the interaction of the various behaviors is often difficult to predict and debug for systems that could perform tasks that require high levels of intelligence. This process is very time-consuming, and undesirable interactions between the behaviors are often difficult to predict and are not always easy to track down. Robust individual behaviors must be designed and implemented, and then tuned to fit the response characteristics of the sensors and effectors.

The motor schema approach to reactive control has proven to be a powerful method in the field of robotics [3]. This model of robotic systems allows researchers to construct robots that can function robustly and in real time in a dynamic, open world. In particular, this method enables the integration of a high-level planner to configure and instantiate these behaviors, thereby introducing more flexibility than is found in the purely reactive approach. A more detailed description of the motor schema method is given in Section III.A.

B. Robot Learning

There are several factors to be considered in designing a robot navigation system that learns. It is desirable that the learning be unsupervised. It would not be practical for a "trainer" to instruct the robot as it moved through the environment, because of the large number of examples that are needed to allow the robot to generalize its navigational knowledge to a wide variety of environment types. Also, there is the issue of credit and blame assignment in navigation. The state of reaching a goal or colliding with an obstacle is achieved through the combination of many simple actions. It is impossible to point to one particular move that led to either good or poor performance. We desire a navigation system that can evaluate its own plans and learn based on some function that specifies the costs and benefits that can be computed from easily-measurable behaviors of the system. For example, the time of travel of a robot from start to goal can be easily and objectively measured, without the need for the designer to examine the internal functioning of the navigation system.

It would be desirable for such a system to be able to learn how to move the robot through an environment without having to go through this process for each type of individual robot environmental configuration. A self-teaching navigation system can improve its performance by using some simple evaluation rules that are provided by the designer. These evaluation rules would be mapped to the behaviors of the robot, in such a way that it can respond robustly to situations that it had not necessarily encountered in the learning phase. This generalization of learning is an important issue in robot navigation, since there are infinitely many possible environmental configurations that can impinge on the sensors.

Although most AI researchers would agree that learning is an important feature of a truly intelligent and autonomous robot system, limited work has been done in this area that goes beyond the conceptual stage. Fikes, Hart, and Nilsson extended the STRIPS robot navigation system to allow it to learn from its experience failures [9]. Researchers have attempted to solve nonlinear robot navigation tasks using two-layer connectionist networks [4]. This simulation allowed the robot to learn associations between landmark and the directions of travel that would lead it to the goal, which would provide positive reinforcement. Previous workers have also applied genetic algorithms (GAs) to robot navigation. Dorigo and Schnepf have used GAs to train simulated robots to avoid obstacles and follow moving targets [6]. In this work, GAs were used to determine when the robot should switch from one behavior to another, and only one behavior is active at any one time. Thus the grain size of the learning is at a fairly high level, and the robots could not learn how to optimize their individual behaviors.

Fig. 1. The genetic operators reproduce(R), crossover(C), and mutate(M) (arranged from top to bottom).

Our research treats the GA learning not as an end in itself, but as a means by which global path planning and local navigation can be coordinated in a more integrative fashion. The global path planner (not implemented in this system) would pass the positions of subgoal landmarks and a measure of the crowdedness of the local environment to the schemas, thus affecting *where* the robot goes and *how carefully* it navigates through the obstacles, without explicitly stating what path to take. This separation of local and global navigation tasks is provided by the schema-based approach, as outlined later in the paper.

C. Genetic Algorithms

Genetic algorithms (GAs) are a hill-climbing search method in which a near-optimal solution may be found by applying a set of operators to points in a search space to produce better "generations" of solutions. The fitness of each member of the GA population (the points in the search space) is computed by an evaluation function that measures how well the individual performs with respect to the task domain. The best members of the population are rewarded proportionately to their fitness, and poorly-performing individuals are punished or removed completely. In this way, the population, and thus some of the individuals, quickly "homes in" on the optimal solution(s) to a problem.

Genetic algorithms depend on an encoding of the domain being learned for the genetic operators to operate on. The encoding is usually in the form of a position-dependent bit string, where each bit is a gene in the string "chromosome." The components of GAs that produce the learning in the system are the genetic operators that are applied to the bit strings of the members of the population. There have been several proposed operators, but the three most frequently used are *reproduction, crossover*, and *mutation.* These operators are expressed graphically in Figure 1. Each of the rectangles in the figure represents a single bit of the string (most representations use much longer strings).

In reproduction operator, the fittest individuals are copied exactly, and replace individuals that are less fit. This increases the ratio of good individuals to the number of poorly-performing individuals. The selection of individuals to be reproduced is done using a weighted roulette wheel selection, so that the best individuals are preferred, but not guaranteed, to be reproduced.

The crossover operation allows two individuals to exchange information by swapping some part of their representation with other individuals. This creates new individuals that may or may not perform better that the parent individuals. The choice of which individuals to crossover and what bits to exchange is done randomly, and it is this random search component that gives GAs much of their power [10].

The mutation operator is used to prevent the loss of information that occurs when there are many nearly-optimal individuals that are missing an important value in their bit string. This operator adds a random factor to the individual members, without affecting the rest of the population. The individuals can "jump" out of local minima and come closer to the optimal value. Mutation must be used carefully, since it often causes near-optimal individuals to perform worse. The mutation rate is decreased linearly during the simulation.

These operators allow the individuals to share information between themselves and improve the fitness of some of the individuals. This is an important point in GA theory; some of the offspring of the genetic operators have a lower fitness than their parents, but on the average the fitness of the population, and the best individuals, improves with successive generations. If the GA is designed and coded properly, it eventually settles on a cluster (or multiple clusters) of near-optimal individuals with similar bit strings. The quality of the solution and length of time it takes to get there depends on the nature of the problem and the values for the many parameters in that control the GA.

III. Task and Approach to the Problem

The navigational control parameters were learned in a two dimensional world that is composed of static virtual obstacles and a single goal position, as shown in Figure 2. The task of each robot in a simulation is to move from the start position, past the obstacles, and to the goal position. All of the simulations used the same distance from start to goal. Simulations were run with varying numbers of obstacles and with varying groupings of fixed numbers of obstacles. A "virtual collision" with an obstacle, which is defined as an intrusion of the robot into the safety margin surrounding the simulated obstacle, is not considered to be lethal, but the fitness of the robot is decremented by an amount determined by the punishment parameter. Robots are rewarded for minimal traversal time from start to goal; the smaller the time, the larger the reward. The size of each step of a robot can vary from zero to some maximum value, and the size depends on values of the various control parameters for that robot and the distance and position of the nearest obstacles and the goal.

Fig. 2. Partially complete robot navigation task in a crowded environment.

A. *Schema-Based Navigation*

The design of a reactive control architecture can be seen as having two parts; a structure and a set of values. The structure is determined by the tasks that the robot must perform, since this constrains the collection of behaviors that the robot can exhibit. Simple robots that are designed to avoid predators need few behaviors, while more complex robots may also have goal seeking and exploratory behaviors. Once the structure of the system has been defined, the system is tuned by adjusting the values of the parameters that make up the behaviors. There are several parameters that control a behavior of a schema-based navigation system, and coming up with a good set of values is currently less scientifically exacting than we would like.

In the Autonomous Robot Architecture (AuRA) motor schemas provide the reactive component of navigation [2]. Instead of planning by predetermining an exact route through the world and then trying to coerce the robot to follow it, motor schemas (behaviors) are selected and instantiated in a manner that enables the robot to interact successfully with unexpected events while striving to satisfy its higher level goals. Motor schemas are manifested as analogs of potential fields [3]. Multiple active schemas are typically present, each producing a velocity vector driving the robot in response to its perceptual stimulus. The individual vectors are summed together and then normalized to fit within the limits of the robot vehicle yielding a single combined velocity for the robot. These vectors are continually updated as new perceptual information arrives with the result being immediate response to any new sensory data.

Some of the schemas we have already developed include:

- **avoid-static-obstacle** - move away from a non-threatening impediment to motion.
- **move-to-goal** - move towards an attractor.
- **move-ahead** - move in a pre-specified compass direction.
- **stay-on-path** - find a path in the environment and stay near its center.
- **noise** - move in a random direction, useful for both exploration and handling problems with local maxima.
- **docking** - move first in a ballistic then controlled motion towards a docking workstation.

- Various **maintain-altitude**, **move-up**, and **move-down** schemas useful for navigation in rough terrain.

B. Simulation Experiments

A robot in our simulation uses three primitive behaviors: *move-to-goal*, *avoid-static-obstacle*, and *noise*, as defined in the previous section. These behaviors define the structure of the navigation system. The three behaviors emerge from a set of five parameters:

- Goal gain - speed at which robot approaches the goal.
- Obstacle gain - speed at which robot moves away from the obstacles.
- Obstacle sphere-of-influence - distance from obstacle at which robot is repelled.
- Noise gain - amplitude of random wandering.
- Noise persistence - number of steps the robot holds the noise vector.

These parameters control the direction and speed at which the robots move through the environment. Because the structure of the system is fixed, a robot's navigation always exhibits the three behaviors with some strength, as long as the values of their gains are not zero. The parameters control the strength of the behaviors, and the system learns one or more sets of parameter values for a given type of environment and a given evaluation function. This optimization is nontrivial, since there is a large set of possible combinations of values, and a large number of these combinations cause the robot to become stuck in areas where the obstacle avoidance force just cancels the force of the goal attraction, especially in regions where box canyons predominate.

C. Genetic Algorithm Methodology

Our implementation of the robot navigation GA uses a floating point representation of the control gains, instead of the standard bit string representation. Although this makes the design of genetic operators more complicated, it does not affect the rate or quality of learning, since an equivalent bit string would just be an encoding of the gains that control the navigation. By using a floating point representation for each of the gain values, the simulation does not need to translate from the bit string to a gain value that can be used by the reactive control system, thus increasing the efficiency of the algorithm.

The algorithm starts by generating a population of robots, an environment of obstacles, and a single goal. Each of the robots has a set of randomly generated values for each of the motor schema parameters. The robots move through the environment until they reach the goal, or until it is obvious that the robots that are not at the goal are "stuck" and are making no progress towards the goal. This constitutes a single simulation. Next, the obstacles in the environment are randomly reconfigured to ensure that the learning is not specific to a particular environment. Another simulation run is made, using the same navigation parameters as in the previous simulation. A running total of the fitness of each of the robots is kept over three simulations, and this value is used during the application of the genetic operators. The top-level code for the algorithm is shown in Figure 3.

```
for(generation=1; generation<MAX_GEN; generation++){
   if ((generation % RECORD_FREQUENCY) == 1)
     /* arrange obstacles as they were originally */
     obstacles.unscramble();

   /* Move robots, see if they reached  */
   /* goal or collided with an obstacle */
   for (step = 1; step < MAX_STEPS; step++)
     robots.moveAndCheck(&obstacles, &goal);

   if ((generation % GA_FREQUENCY) == 0){
     /* Apply the GA operators. */
     robots.reproduce(P_REPRODUCE);
     robots.crossover(P_CROSSOVER);
     robots.mutate(P_MUTATE);
   }
   /* randomize obstacle positions */
   obstacles.scramble();
 }
```

Fig. 3. Top-level code for the genetic algorithm simulation.

The evaluation of the robots that is performed during the simulation is central to the GA algorithm, since it is this evaluation that drives what the robots learn. There are several "goodness of path" measures that one may want to optimize. Of primary concern is speed, since we usually want our robots to perform a task in minimal time. Robots that take the least number of time steps to reach the goal are given the highest fitness values, while those that collided with obstacles are penalized and given low fitness ratings. Also important is the need to minimize the distance traveled and the energy expended, since autonomous systems are constrained by limited resources. If we have a good model of the robot that the simulations are modeling, we can also take into account the wear on the robot, the safety of the route, and any difficulties that it may have navigating over certain types of terrain.

The evaluation function used in these experiments begins with a base value, which is the same for all of the robots. For each robot, this base value is decremented by a function based on the number of moves (and thus the time it takes) for the robot to reach the goal. This value is then decremented by a percentage of its value for each virtual collision the robot has with an obstacle. By adjusting these reward and punishment parameters, one can get fundamentally different navigation behaviors to emerge. Low punishment values produce robots that tend to crash into the virtual obstacles frequently, but get to the goal more quickly. The robots that result from high punishment values are more passive and take a safer route, while taking longer to reach the goal.

Once fitness ratings have been calculated over several navigation simulations, the genetic operators are applied to the population. The fittest individuals are reproduced, and these

```
fitness(x) = max_fitness - time_to_goal(x) -
             (collision-weight * collisions(x))

WHERE:
  max_fitness = the maximum possible fitness.
  time_to_goal(x) = number of steps robot x took.
  collision_weight = penalty for each collision.
  collisions(x) = number collisions for robot x.
```

Fig. 4. One possible evaluation function for measuring performance of a robot.

offspring replace the least fit robots. Then the crossover operation is applied to the population, and again the fittest individuals have the highest probability of combining with other individuals to create robots with new parameter sets. Finally, individuals are randomly chosen from the population to be mutated, and the parameter sets of these robots are changed slightly.

The process is then repeated; the population, with the new and mutated robots, is allowed to move through the environment and the individuals are assigned new fitness values. How many times this is repeated depends on several variables; the density of the obstacles, the size of the population, the amount of optimality desired, and the closeness of the initial random control parameters all affect the amount of processing required.

IV. Results

Computer simulations were run to study the behavior of our system, and to evaluate the magnitude of improvement that the GAs can provide for the navigation tasks.

A. Factors Affecting Learning

One factor that determines what the robots actually learn is the type of environment that they are learning in. Environments can vary along several dimensions. The density of the obstacles in the environment affects how much the "best" route deviates from a straight line from the start to the goal, and how frequently the robot has to zig-zag through the obstacles. As the density of the environment increases, it becomes more efficient to go around a cluster of obstacles rather than try to navigate through them (and risk a possible collision). Environments may also vary in the organization of the obstacles; they may be totally random, be clustered into groups, or form highly-organized patterns such as fences or box canyons. One may also add the complexity of moving obstacles and goals, each of which can have their own peculiar behavior. The type of navigation that is optimal varies between these environments, and the robots should learn these different types of navigation behaviors so that they may be used in similar future situations. Of course, the robot must also learn how to classify an environment, and possibly interpolate between known environments, to be able to use this information effectively.

Robots that learn to navigate in a particular environment are optimized to navigate in similar environments. Thus a simulation for that environment forms a "niche" in the same way that animals evolve to fill a particular niche in nature. Take such an animal outside of of its evolutionary niche and it performs poorly, but left in its environment it does just fine. One could set up a set of simulations to form several niches, and then design a high level controller to choose the niche parameters that are right for a particular world. Thus the reactive control system would adapt to its current environment, making it more efficient and more robust. Although this is beyond the scope of this paper, it does point to one interesting direction for future work.

Since the fitness function is driving the learning, it also affects what is learned. Robots that learn with a large punishment for collisions take the safe route, but may no longer be able to navigate between certain objects. On the other hand, robots with no collision punishment can navigate through most environments more quickly than the punished robots, but often collide with obstacles. The amount of punishment that one assigns to a collision should depend on the robustness and strength of the robot being simulated.

B. Evaluation

The evaluation of GAs as applied to robotic navigation learning can be done in several ways. Each of these methods tells something about the learning of the robots, but does not tell the whole story. This section describes three methods that were used to evaluate these simulations. One of these evaluation methods is included in each of the three simulated environments described later.

1) Objective behavior measures: The simplest method of evaluation would be to measure the change of the fitness values for the robots over several generations. If the simulation is working correctly, the robot population shows an increase in the average value over the course of a simulation, with a convergence toward an optimal value at the end of the simulation. However, there are problems with using this method. It is this fitness value that is driving the learning, and this function is only a qualitative measure of what a good path would be like. An increase in the quality of the path occurs only if the evaluation function accurately models the interaction between the robot and the real world. It is possible for the population to show an increase in fitness value, without a corresponding increase in the quality of the path. We need a criteria for testing the success or failure of the learning algorithm that is external to the operation of the GA.

One evaluation method we use is to analyze the objective measures that the fitness values were calculated from to evaluate the learning of the robots. Our simulation uses time of travel and number of collisions to calculate the fitness values, and we analyzed these values to ensure that the robots were learning what was intended. For example, when a large punishment was used for collisions, the number of the collisions of the robots quickly dropped to zero, while causing an increase in the total number of steps needed to get to the goal.

2) Visualization of the navigation: Although the use of visualization methods to evaluate the learning algorithm is admittedly subjective, such methods can provide many insights into what the robots are learning. Also, using purely objective measures of performance do not tell us much about what kind of behaviors the robot is likely to exhibit. In an open world there is a tradeoff between length of path and safety, and looking only at the path length leads to systems that ignore the issues of safety and robustness of navigation. By using a visualization of the simulation, one can get a qualitative feel for the success of the learning algorithm.

The GA simulations produce a trace of the moves of the robots for each run. A separate program is then used to display the movements of the robots for each of the simulated navigations. This program displays both the first generation and one of the later generations on the same screen, so that the user can compare the two simulations and judge the change in behavior for the later generations. To insure that the outcome of a displayed simulation is the result of the learning and not of the particular environment, the configuration of the obstacles is reset to that of the first generation when a particular simulation run is saving the moves of the robots for viewing.

3) Convergence evaluations: The third method used in evaluation is to look at the navigational parameters and determine whether they converge to one or more sets of values. The gain values at the start of a simulation are randomly generated, and are fairly evenly distributed over the range of possible initial values. If there is an optimal solution to the problem, given a simulation environment and fitness function, then the population converges toward that value. The number of generations that it takes to converge depends on the GA parameters and the difficulty of the problem.

In each of the experiments, a trace of the values for the five reactive control parameters was saved for later analysis. Given enough time to converge (usually around 400 generation with a population size of 30), each of the reactive control parameters of the populations converged to within about 2 percent of the starting range for that parameter. It is possible that some environmental configurations may result in several subpopulations with similar fitness values and different reactive control values, but no such environments were found in these experiments.

C. Simulation Results

Simulations were run on three types of environment of increasing complexity. The *sparse* environment contained 50 obstacles in the field, and these obstacles were not organized into higher-level structures. The *crowded* environment contained 500 obstacles, and presented a more difficult learning problem for the robots. The third and most difficult environment was the *box canyon* problem, which had the obstacles arranged so as to "trap" the robots. The results of these simulations are presented below.

1) A sparse environment: This environment has only 50 obstacles, and there are several possible paths through the environment that are very nearly the same length. In the first generation, some of the robots are too slow to get to the goal in the allocated time. The noise component that is fairly large in the first few generations settles down to a value that is more reasonable for this type of environment. Since there are few local minima/maxima in this environment, there is little need for noise to push the robots around the obstacles. The robots learn that noise is not important in a sparse environment, and, as the value of the noise gain decreases, the performance of the robots increases.

Fig. 5. Evolution of time and collisions measures for a sparse environment (each box represents the range of values, the middle bar is the median).

Figure 5 shows the median, minimum, and maximum values for the number of steps and the number of collisions for several generations of robots, with the median values as the dark band inside the boxes, and the minimum and maximum as the tops and bottoms of the boxes, respectively. The dotted line at the bottom of the graph is a benchmark used to judge performance for both the number of steps and number of collisions. The distance from the start to goal positions (the absolute minimum for a path *with no obstacles* in the environment) is 400 steps, and we would like to have no collisions for this simulation. The evaluation function decreased the fitness of the robots by 5 percent for each collision. By the time the simulations reaches the 240th generation, there are no collisions, and all of the robots reach their goals (each robot was allowed 1200 steps to travel to the goal).

None of the robots in this simulation could minimize the number of steps to the number of steps on straight line between the start and goal positions, since there were several obstacles directly between the start and goal. Also, it is not clear whether is is desirable to optimize this measure to the point that the robot takes sharp turns at the obstacles. In real-world navigation, obstacles may move unpredictably, sensors are noisy, and the robot locomotion system may not go exactly where it is told to. This is one of the advantages to this approach; it gives a "margin of safety" that is useful in unpredictable environments. The robots learn to navigate through the environment at a distance from the obstacles that will prevent collisions, as can be seen from the convergence of the number of collisions to zero in the above figure.

Fig. 6. Range of values for the *obstacle sphere-of-influence* and *noise gain* parameters over several generations in a crowded environment.

2) A crowded environment: This environment for this simulation contained 500 obstacles, and also had a fence-like arrangement of obstacles in the middle of the field. The environment is much harder to learn how to navigate through than the sparse environment, since the spaces between obstacles is smaller and there is a greater chance of collision. This higher probability of collision would favor low noise values, but the greater chance of being stuck in a local minimum would favor high noise values. Also, there is tradeoff between high obstacle gains with small spheres-of-influence, and low gains with large spheres-of-influence. In some situations, it may be better to go around a cluster of obstacles rather than try to navigate through them. Take the example of walking through a room full of chairs; it is sometimes better to walk around the chairs at a fast pace than to walk slowly through the middle of them.

Figure 6 shows the convergence of the robots' *obstacle sphere-of-influence* and *noise gain* parameters. The initial parameter settings are random across the population, as can be seen by the distribution of the values for the first generation. Over successive generations, the values of these parameters converge toward a small range. One interesting feature of this simulation is that the final value for the obstacle sphere-of-influence parameter is outside the range of the values of the first generation. The algorithm is able to find a value that is not an interpolation of the initial set of initial values. None of the simulations in this experiment showed sensitivity to initial values, given a large enough population size and number of generations.

3) Box canyons: This type of environment is difficult for a purely reactive control system to navigate in, since the robot must first move away from the goal when it becomes trapped in one of the canyons. In some cases the noise component knocks the robot out of valley, but one can design a environment configuration that "tricks" the robot into being trapped for some amount of time. These non-linear problems usually require knowledge about the meta-configurations of obstacles, and procedures that can be use to get out of the trap and move toward the goal. The navigator of a hierarchical planner must be able to determine when the robot is not making progress toward the goal, and indicate a new direction for the reactive control component.

Figure 7 contains several frames from the visualization program for the box canyon environment. In the figure we can see the navigational behaviors, from moving left to right, of 30 robots in generations 1, 100, 300, and 500 (from top to bottom). Although these pictures do not capture the dynamics of the navigation, they do represent the navigational behaviors of the robots. The dark "fuzzy" areas are regions where the repulsion from the obstacles is too high to allow the robot to pass. This happens both where the obstacles are close together, in which case the robot can pass between the obstacles, and in the box canyons, where the robot cannot pass through and must go around. In each successive generation, more of the robots are able to reach the goal, although even in the last generation some of the robots still become trapped. Also, notice the increase in the distance from the paths to the obstacles, as the robots learn to avoid collisions and box canyons. The average path length of the last generation was slightly greater than for the first generation, since the evaluation function used a high punishment value for the virtual collisions, but the average path completion time (number of steps) was significantly less.

This simulation shows the interaction between the navigation parameters. Large obstacle repulsion gains prevent the robot from colliding, but do not allow the robot to pass between two obstacles that are close together. Likewise, noise allows the robot to jump out of local minima, but increases the travel time. For complicated environments with great distances between the start and goal, one set of reactive control parameters does not guarantee that the robot can navigate to the goal.

V. Conclusion

Genetic algorithms provide a powerful method for searching for near-optimal solutions in complex search spaces. Drawing from analogies in biology and evolution, they can be applied to the learning of robotic coordination in reactive control systems. It is hoped that many of the behavioral parameters that are presently being programmed into these systems manually can instead be learned by systems similar to the one presented in this paper. This work shows that GAs can be used to learn such parameter values for reactive control navigation, and that simulated systems that use these learned parameters perform well in similar environments. An important focus for future work is the integration of the learned parameters into a hierarchical navigation system.

Fig. 7. Navigation (from left to right) of 30 robots for in a box-canyon environment for generations 1, 100, 300, and 500 (top to bottom).

References

[1] J. Albus, H. McCain, and R. Lumia. NBS Standard Reference Model for Telerobot Control System Architecture (NASREM). NBS Technical Note 1235, Robot Systems Division, National Bureau of Standards, Washington, D.C. 1987.

[2] Arkin, R., Riseman, E. and Hanson, A., AuRA: An Architecture for Vision-based robot Navigation, *Proc. DARPA Image Understanding Workshop*, pp. 417-431, Los Angeles, Feb. 1987.

[3] R. Arkin. Motor Schema-Based Mobile Robot Navigation. *The International Journal of Robotics Research* Vol.

[4] A. Barto, C. Anderson, and R. Sutton. Synthesis of Nonlinear Control Surfaces by a Layered Associative Search Network. *Biological Cybernetics* Vol. 43, pp. 175-85. 1982.

[5] R. Brooks. The Whole Iguana. *Robotics Science*, pp. 432-56, 1989.

[6] M. Dorigo and U. Schnepf. Organization of Robot Behavior Through Genetic Learning Processes. Submitted for publication, 1991.

[7] E. Dickmanns and A. Zapp. Guiding Land Vehicles along Roadways by Computer Vision. *AFCET Conference* Toulouse, France. 1985.

[8] A. Elfes. Sonar-based Real-world Mapping and Navigation. *IEEE Journal of Robotics and Automation.* R-A-3(3), pp. 249-265. 1987.

[9] R. Fikes, P. Hart, and Nils Nilsson. Learning and Executing Generalized Robot Plans. *Artificial Intelligence* Vol. 3, pp. 251-88.

[10] D. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Publishing Company, 1989.

[11] L. Kaelbling. An Architecture for Intelligent Reactive Systems. *SRI Technical Note* V 400, SRI International. Oct. 1986.

[12] D. Payton. An Architecture for Reflexive Autonomous Vehicle Control. *IEEE Conference on Robotics and Automation.* pp. 1838-1845. 1986.

[13] A. Petland and R. Bolles. Learning and Recognition in Natural Environments. *Robotics Science*, pp. 164-207, 1989.