Genetic Programming of Autonomous Agents for Perimeter Maintenance

Scott M. O'Dell
Bradley University
1729 Churchill Dr.
Dubuque, IA 52001
1-563-581-8713
scott.odell.m@gmail.com

Joel D. Schipper
Bradley University
1501 West Bradley Avenue
Peoria, IL 61625
1-309-677-2260
jschipper@bradley.edu

ABSTRACT

Research in genetic programming (GP) for the control of autonomous agents often employs grid-based simulations to evolve prototypical solutions. However, grid-based simulations produce solutions that are often impractical on physical robotic platforms. Our research evolves perimeter maintenance control programs to illustrate the limitations of grid-based approaches and explore the advantages of using a continuous simulator when evolving programs for physical autonomous agents.

Perimeter maintenance is a task where a group of autonomous "guard" agents are deployed around a "base" to detect "enemy" agents before they reach the base. For this research, GP software and simulators were developed to approximate the movements of physical agents. To illustrate the limitations of evolving autonomous agent control programs using a grid-based simulator, early experiments evolved perimeter maintenance agents in the grid domain. Later experiments replaced the grid-based simulator with a continuous simulator to demonstrate its advantages when generating control programs for physical autonomous agents. The experiments performed explore multiple fitness functions for evolving homogenous and heterogeneous teams of guards as well as the co-evolution of guards and enemies. The approach has produced autonomous agent control programs that display intelligent perimeter maintenance behavior in a continuous simulator by patrolling a circular area around the base.

Keywords

Genetic Programming, Autonomous Agents, Perimeter Maintenance, Cooperative Multi-Agent Systems, Co-Evolution.

1. INTRODUCTION

Genetic programming (GP) is a machine learning technique based on the work of John R. Koza [1], with origins rooted in genetic algorithm theory [2]. By simulating natural selection, GP can evolve programs that solve complex problems with very little input from the designer.

GP begins by producing a *generation* of random programs (i.e. genomes) composed of elements from a designer specified *primitive set*. The primitive set must be chosen to give the program sufficient perceptual, computational, and locomotive ability to effectively perform its task. Primitives that require arguments comprise the *function set,* while primitives without arguments comprise the *terminal set*.

GP uses a fitness function to evaluate how well each randomly generated program performs a designer specified task. A fitness function returns each program's *fitness score*. Programs with higher fitness scores are more likely to contribute their genetic material to the next generation. Fitness functions must reliably separate more fit from less fit individuals, even if all individuals are relatively unfit.

GP produces successive generations of programs using *genetic operators*. First, members of the current generation are selected in proportion to their fitness. Genetic operators then simulate sexual reproduction (crossover) by combining portions of two programs to produce a new program, asexual reproduction (reproduction) by making an exact copy of a program, and biological mutation (mutation) by randomly altering a portion of the program. These genetic operators produce a new generation that behaves similarly to the fittest individuals from the previous generation, focusing GP's search for programs on structures that demonstrate high fitness. The fitness function evaluates the next generation of programs which in turn is used to produce yet another generation. This process of evaluation and procreation repeats until a specified number of generations is produced or a specified fitness level is met. The output of a genetic programming sequence is the fittest individual produced during any generation.

The solutions generated by GP are sensitive to evolutionary factors such as the physical accuracy of simulated environments, the genetic relatedness of cooperative programs, and the presence of evolving opponents. In this research, the effects of these factors are investigated by evolving autonomous agent control programs to perform perimeter maintenance behavior. The results are then evaluated to determine how each factor contributes to the creation of programs that are appropriate for deployment on physical autonomous agents.

2. LITERATURE REVIEW

2.1. Perimeter Maintenance

Perimeter maintenance is a practical military application useful for defending a base. As illustrated in Figure 1, a team of autonomous "guard" agents (green) cooperate to detect intrusions into the perimeter (blue) surrounding a "base" (black) by "enemy" agents (red). In [3], engineers use emergent behavior principals to develop perimeter maintenance robots. Sensor data representing the positions of nearby objects and the base is sufficient to produce perimeter maintenance behavior. The robots are designed to accept a perimeter size as input and revolve around the base at that distance. The percentage of the perimeter monitored is used to assess the quality of the control program. Simulations of the robots were run in the MobileSim simulator, which does not account for the noise that occurs in physical systems. When the control programs were placed on physical robots, the robots did

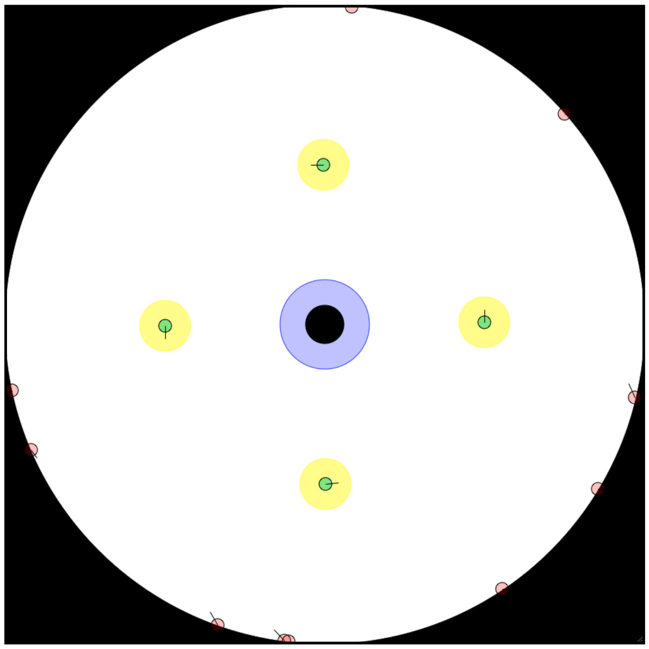

Figure 1. Perimeter Maintenance Example

not behave as simulated due to a noisy environment and sensor readings. To create control programs that operate on physical robots, the development simulator must include realistic sensor noise. Our research evolves agents that perform perimeter maintenance because it is an excellent application for testing practical autonomous agent control programs due to its reliance on spatial properties of the environment.

2.2. GP for Autonomous Agent Control

Research utilizing GP to evolve autonomous agent control programs traditionally simplifies the environment in which solutions are evolved to quickly produce prototypical solutions that support a thesis. A commonly used simplification is employing a grid-based simulator for the world model [1], [4].

Simplifying the simulated environment has a dramatic impact on the solutions GP produces. Using grid-based simulators to evaluate fitness produces control programs that rely on discrete movements and are difficult to adapt to physical autonomous agents in the continuous real world. The results of our research indicate that grid-based simulators also distort the space they represent, making curved navigational paths inefficient.

2.3. Evolving in a Complex Environment

For evolutionary agents to develop useful reactions within an environment, information about the environment must be available in a suitable form. Luke [5] uses genetic programming to develop a team of agents to play soccer in a noisy simulator designed to simulate soccer matches for the RoboCup software competition. The simulator provides data about the environment that is not directly useful for evolving strategies, which means the time to develop a fit solution is prohibitively large. Luke solves the problem by condensing complex environmental data and agent actions (i.e. intercepting a soccer ball) into elements in the primitive set. The complex primitives allow the evolutionary sequence to focus on developing soccer-playing strategies rather than data processing routines. Although writing complex procedures for the primitive set hastens the evolution process, it results in blocks of immutable code that cannot be optimized by the GP process. Therefore, this method is not pursued in our research. However, the concept of simulator noise is utilized in later stages of research to evolve agents that can account for uncertainty in the environment.

2.4. Cooperative Agents and Co-Evolution

Autonomous agents in genetic programming can evolve to cooperate explicitly through data exchange or implicitly through interaction in an environment. The effectiveness of cooperation depends on team composition and the task to be accomplished. In [6], Floreano and Keller find that the evolved solutions for cooperative agents in genetically homogenous versus heterogenous teams are directly dependent on the nature of the fitness function. Homogenous teams produce superior results with fitness functions that require high-cost cooperation, whereas heterogenous teams are superior for evolving low-cost cooperation. Koza [1] finds that co-evolving opposing agents often produces more robust results then GP with a single population. Our research examines the effects of homogenous versus heterogenous team selection and co-evolution in creating practical autonomous agent control programs.

3. GP FRAMEWORK

The GP software developed during this research is based on the template presented in [7] and written in the Ruby programming language. Ruby was selected for its rapid development cycle and the ease in which it can interface with faster languages if simulation time becomes an issue.

Figure 2 displays the architecture of the software components that comprise the GP framework. To evolve programs, a function set, a terminal set, and reproduction parameters are provided to the genetic programming evolutionary sequence (GPES) object,

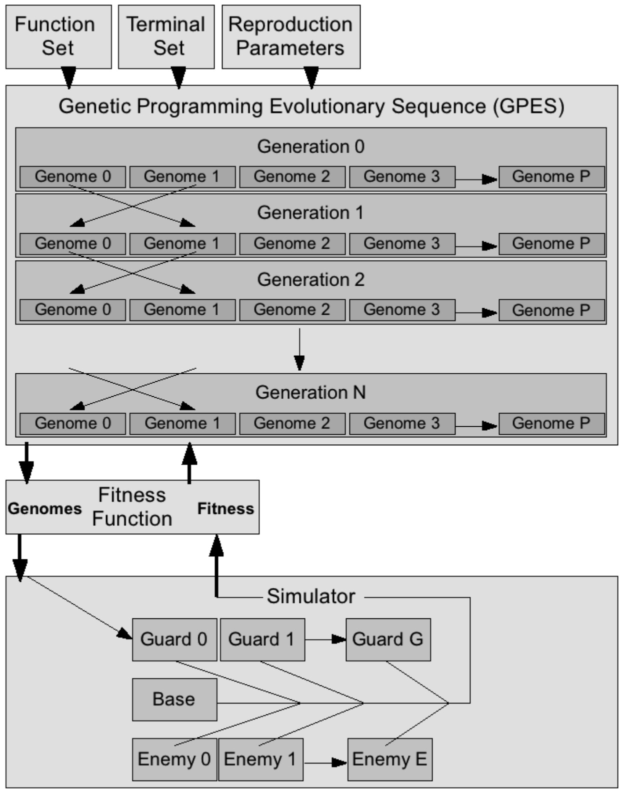

Figure 2. Organization of GP Framework

which organizes the creation of generation objects. Each generation object has methods to create and organize the genome objects that store programs as nested arrays. The GPES object passes the randomly produced genomes of the first generation to the fitness function object, which initializes the simulator. Within the simulator, the genomes act as control programs for guard agents. The interactions between guard, enemy, and base agents within the simulator determine the fitness score of the genome. The fitness function object passes the fitness scores back to the GPES, where a new generation initializes genomes by performing genetic operations on members of the previous generation. This process continues until the final generation is evaluated for fitness. The result returned by the GPES is the genome that achieves the highest fitness score.

4. GRID-BASED SIMULATIONS

Initial attempts to produce perimeter maintenance behavior used GP to evolve autonomous agent control programs in a grid-based simulator. The results of these experiments demonstrate the limitations of grid-based simulations when evaluating the fitness of autonomous agent control programs.

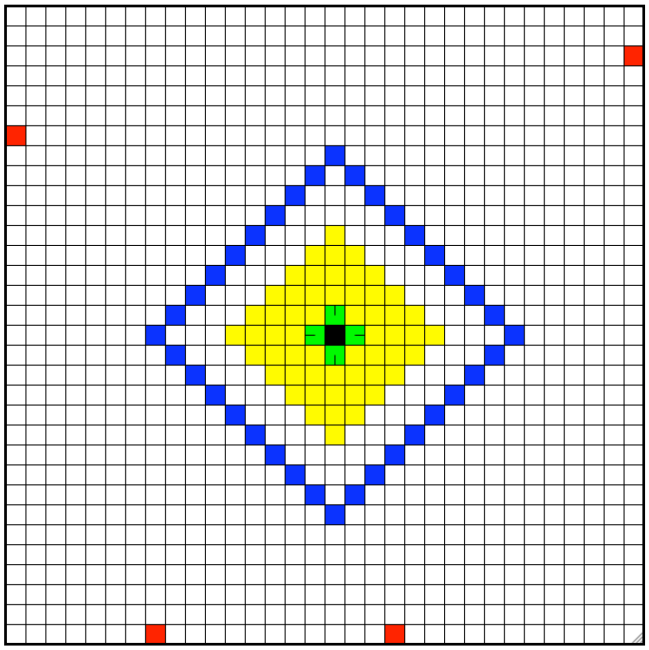

Figure 3. Starting Positions in the Grid-Based Simulator

This research implements a grid-based simulator with a 32 by 32 unit grid in which each agent occupies a single square unit. Figure 3 illustrates the structure of the simulator. A base agent (black) and four guard agents (green) begin at the center of the grid. Enemies (red) enter the simulation at random times and in random locations on the edge of the grid. The desired perimeter size is measured using the Manhattan distance from the base (blue). When an enemy enters a guard's sensor range (yellow), that enemy is removed from the simulation and the guard's fitness score is updated to reflect the successful detection of an enemy. If an enemy enters the base's perimeter, the enemy is removed without affecting the guard's fitness score. Grid-based simulations utilize two fitness functions. The *quantitative fitness function* bases a genome's fitness score on the number of enemies detected during the simulations. The *qualitative fitness function* adds the Manhattan distance between the enemy and base to the fitness score each time an enemy is detected, thus, rewarding guard agents for detecting enemies further from the base.

4.1. Primitive Set

For guard agents to effectively perform perimeter maintenance, the primitive set must allow the guard to conditionally execute code depending on its distance from the base and move around the simulator to detect enemies. All elements in the primitive set accept and return integer values.

Table 1. Function Set for Grid-Based Simulation

Function	Arity	Pseudo-code
prog	2	(a) then return(b)
>	4	if (a > b) then (c) else (d)
+, -, *, /, %	2	[standard integer arithmetic]

Table 1 outlines the operations performed by the function set. The "prog" function allows the control program to perform a series of actions by evaluating the two arguments sequentially. The ">" function gives the program the ability to conditionally execute the third or fourth argument depending on the values of the first two arguments. The standard arithmetic set (+, -, *, /, %) enables the control program to apply weights to inputs and derive integer values that are not included in the terminal set.

Table 2. Terminal Set for Grid-Based Simulations

Terminal	Effect
base	returns Manhattan distance from guard to base
forward	moves agent forward, returns "base"
left	turns agent left 90 degrees, returns "base"
right	turns agent right 90 degrees, returns "base"
0-6	constant integers

Table 2 explains the values returned by the terminal set. The "base" terminal returns the Manhattan distance to the base, which can be used to conditionally execute procedures based on that distance. Terminals "forward", "left", and "right" enable the agent to move to any location on the grid while returning the same value as "base" since the terminals have no value directly associated with them. Terminals "0" through"6" are constant integers that are chosen during the creation of the control program and are inheritable by future generations.

4.2. Homogenous Teams

In the first evolutionary sequence performed with the grid-based simulator, all four guards shared identical control programs. The quantitative fitness function was used with a guard sensor radius of four units and a base perimeter radius of nine units. These sizes make it impossible for the team of guards to fully protect the entire perimeter at any time, forcing them to find behaviors that best defend the base.

Figure 4 shows the result of the genetic programming evolutionary sequence. The evolved guards move to the position shown in the figure and remain stationary for the rest of the simulation. Although this result detects most of the enemies

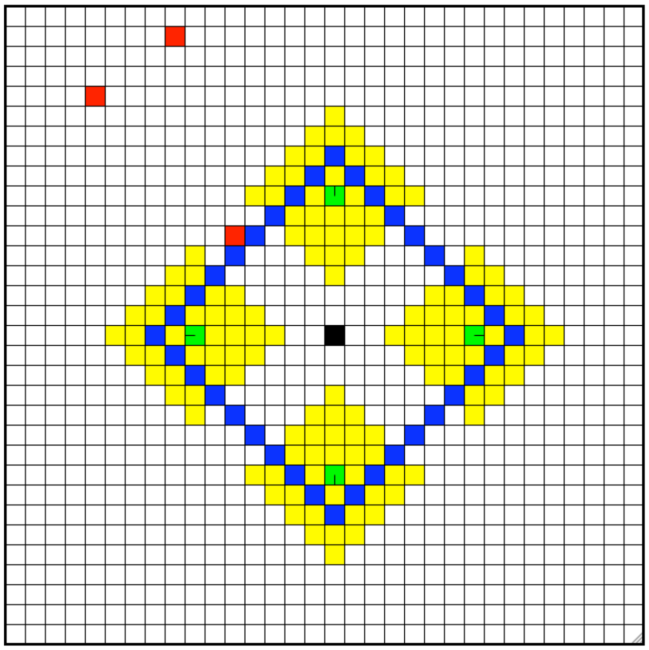
Figure 4. Grid-Based Result for Homogenous Teams

during the simulation, common sense would suggest that a real autonomous agent could protect more of the base by patrolling the perimeter in a circular pattern. The structure of the grid domain inherently prevents the evolution of such a behavior because as a guard moves from one corner of the perimeter to another it leaves a large portion of the perimeter unprotected. The grid domain is a distortion of the real world, which creates positions of higher value that do not exist in the continuous domain. In this case, using a grid-based simulator compromises the practicality of the solution.

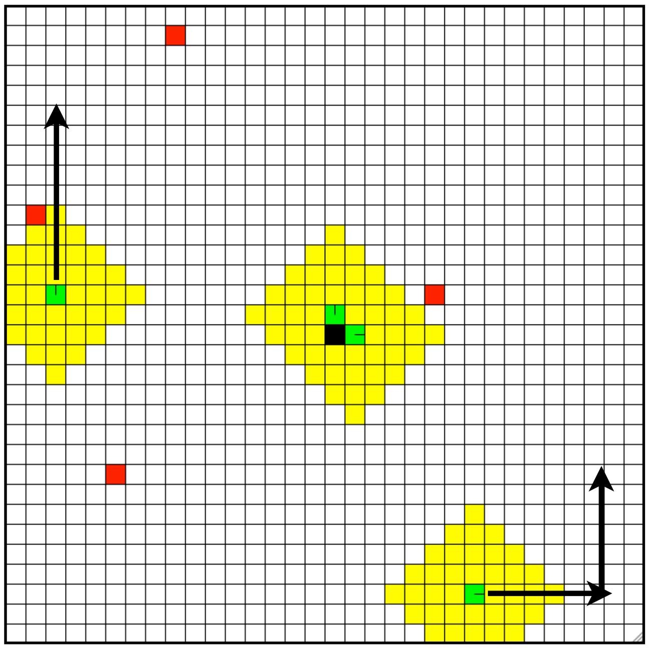
Figure 5. Grid-Based Result for Heterogenous Teams

4.3. Heterogenous Teams

The second evolutionary sequence split the guards into four separate populations to evolve independently so that each population could potentially assume a unique role in defending the base. Each of the four populations had a designated starting position near the base (north, south, east, west). The qualitative fitness function was used for this sequence and the base's perimeter was reduced to a one unit radius. The fitness function took a genome from each of the populations and places it in its designated starting position. Each individual received the fitness score earned by the team as a whole and the process was repeated so that each genome was evaluated with five different teams.

Figure 5 shows the result of evolving heterogenous teams. The guards in the north and east positions remain stationary to ensure that no enemies reach the base and that modest detection scores are added to the team's fitness. The other two positions evolved to seek large detection scores by wandering further from the base, but attaining a lower detection rate. The same effect would be achieved if only one agent remained at the base, but the GP sequence yielded two base protecting programs. This demonstrates the problem of redundancy when evolving heterogenous teams. In populations that remain at the base, genetic operators occasionally produce programs that leave the base. Two populations evolve the same role to ensure the base does not become unprotected. In this case, the nature of GP compromises the practicality of the solution by evolving redundant agents.

4.4. Homogenous Teams with Co-Evolution

For the next evolutionary sequence, a population of enemy agent control programs was evolved simultaneously alongside the guard population. Co-evolution can decrease the predictability of solutions because the opposing population exploits strategic weaknesses until they are corrected. The enemies were given the same primitive set as the guards with the addition of two terminals that return the horizontal and vertical distance to the base relative to the enemy. The quantitative function was used for the guards

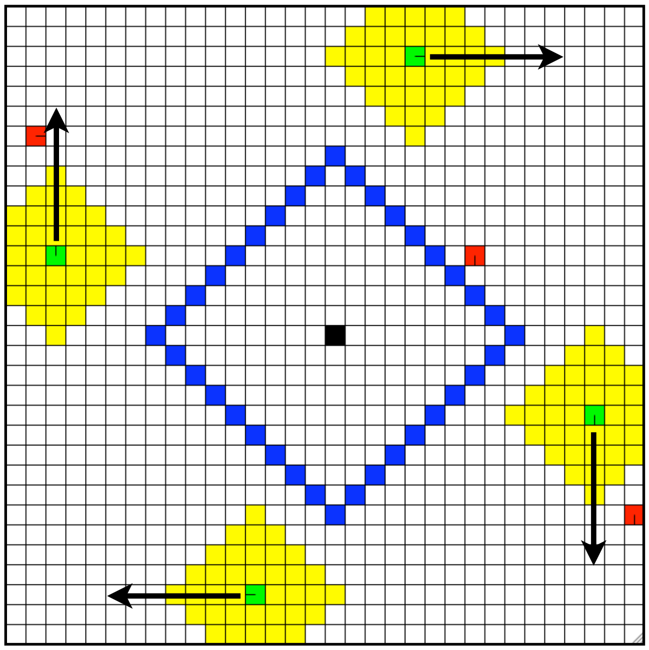
Figure 6. Grid-Based Result for Homogenous Co-Evolution

with a base perimeter of nine units. Enemies operated as homogenous teams and received a fitness score equal to the number of enemies that reached the perimeter of the base.

Figure 6 shows the result of co-evolution using homogenous teams of guards. The guards evolved to patrol the edge of the grid where the enemies enter the simulation. The guards' tactic is the result of the enemies' primitive set enabling them to determine the location of each guard. This makes it unlikely for guards to detect enemies if the enemies have room to maneuver around them. Therefore, the guards attempt to detect the enemies before they have a chance to move. Although the solution displays intelligence, it is not practical because the edge of the grid is a construct of the simulation and has no physical analog.

4.5. Heterogenous Teams with Co-Evolution

During the final grid-based evolutionary sequence, enemies evolved simultaneously with the guards as in the previous sequence. The guards were placed in heterogenous teams where each starting position evolved independently. The quantitative fitness function was used and the base perimeter had a nine unit radius.

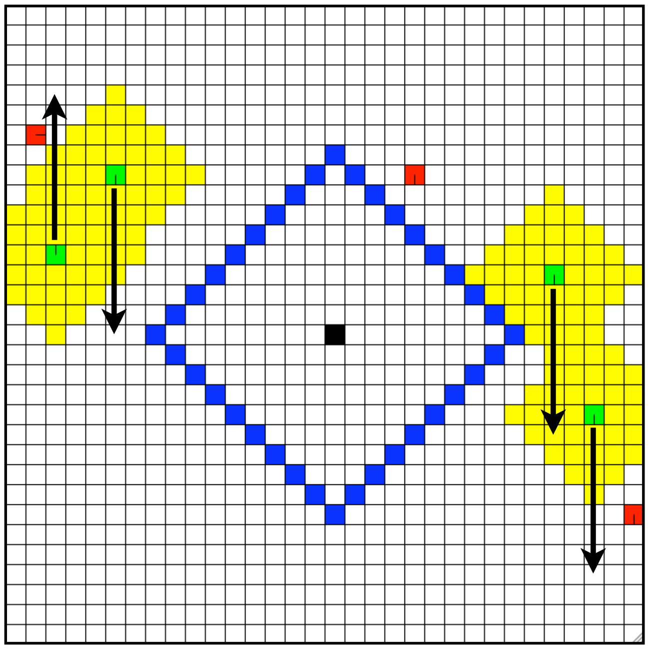
Figure 7. Grid-Based Result for Heterogenous Co-Evolution

Figure 7 shows that the guards evolved a similar strategy to the previous sequence. Each guard moves to the edge of the grid to detect enemies before the enemies can move. Although the heterogenous team makes the movements of the guards more unpredictable, the solution is still not practical because the edge of the grid has no physical analog.

4.6. Limitations of Grid-Based Simulations

The results of using a grid-based simulator show that GP can evolve autonomous agent control programs that display intelligent perimeter maintenance behavior. However, simplifying the world model creates distortions that do not exist in the real world. GP exploits these distorted areas to increase the fitness score of the control programs at the expense of their real-world practicality. GP also exploits the finite size of the simulation and the predictable starting locations of the enemies by patrolling the edge of the simulator.

5. CONTINUOUS SIMULATIONS

The goal of using continuous simulators to develop autonomous agent control programs is to better align the fitness score with practical solutions that accurately represent the movements of physical autonomous agents. The results show that GP can utilize a simple primitive set to produce complex, practical control programs with continuous movement.

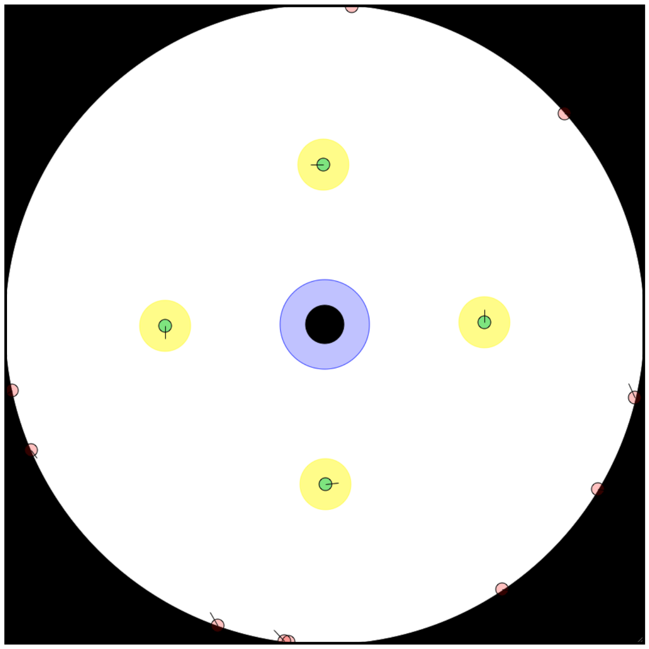
Figure 8. Starting Positions in the Continuous Simulator

The continuous simulator developed for these experiments uses polar vectors to represent agents' positions, headings, and movements. Figure 8 shows the guards in their starting positions, the base in the center, and the enemies starting at random locations at the edge of the simulator. The enemies enter the simulation at random times, so the guards must evolve a general solution to detect them. The quantitative fitness function was used for all experiments in the continuous domain. The simulator has a radius of fifty units, the base perimeter is seven units and the guard sensor range is four units. These values ensure the guards cannot defend the entire perimeter at any given time. Table 3 displays the additional properties of the base, guard, and enemy agents.

Table 3. Properties of Agents

Agent	Radius	Max Velocity	Max Rotational Velocity
Base	3	-	-
Guard	1	1.5 units per second	45° per second
Enemy	1	1.0 units per second	45° per second

5.1. Primitive Set

The primitive set must be simple so that GP can evolve creative solutions that are applicable to physical autonomous agents. Furthermore, it must be robust enough to create controllers for agents moving in continuous space. To operate in the continuous

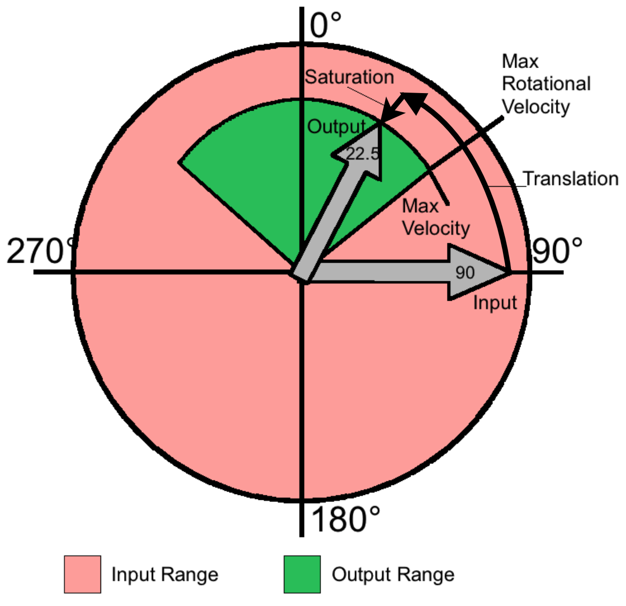

Figure 9. Translating Vectors to Movement

domain, all primitives accept and return polar vectors. The vector returned by the control program after execution is used to determine the heading of the agent during each 150ms time step. Figure 9 shows how the vector returned by the controller translates to a movement vector that is limited by the maximum velocity and rotational velocity of the agent. For example, assume the guard has a maximum velocity of 1.5 units per second and a maximum rotational velocity of 45° per second. If the guard control program returns the vector (3∠90°), the magnitude is saturated to 1.5 and the angle is scaled by 45/180. The resulting movement vector would be (1.5∠22.5°), which would result in forward movement of 1.5 units, and adjustment in direction of 22.5° to the right.

Table 4. Function Set for Continuous Simulations

Function	Arity	Pseudo-code
prog	2	(a) then return (b)
> (Magnitude)	4	if (a.mag > b.mag) then (c) else (d)
> (Angle)	4	if (a.ang > b.ang) then (c) else (d)
X =	1	variable X = (a)
+, -. *	2	[standard vector arithmetic]

Table 4 outlines the function set. The "prog" function is included to allow control programs to calculate and save values without the values directly influencing the heading of the agent. The ">" function is defined for magnitudes and angles so both values can be used to selectively execute control code. The "X=" function allows control programs to store three vectors for computing more complex actions than a typical state-machine. Additionally, the control programs can scale vectors to calculate a new heading using the standard vector arithmetic primitives.

Table 5 shows the members of the terminal set. The "base" terminal returns a vector representing the location of the base relative to the guard, enabling guards to behave differently depending on location. The "direction" terminal returns the guard's orientation (north = 0°, east = 90°, etc.) Terminal "X" returns the value stored by the corresponding "X=" function. Finally, the "vector" terminal represents the static vectors (magnitude from 0-50 and angle from 0°-360°) inserted into the genome at creation so that control programs can make comparisons to vector constants.

Table 5. Terminal Set for Continuous Simulations

Terminal	Effect
base	returns vector from guard to base
direction	returns unit vector representing guard's heading
X	return (variable X)
Vector	[static vector]

5.2. Homogenous Teams

For the first evolutionary sequence in the continuous domain, all four guards in a homogenous team shared the same control program and attempted to detect enemies that headed directly toward the base. Initial solutions for homogenous teams produced guards that remained in their starting position for the entire simulation. Such evolution occurred because most programs that moved the guards caused them to hit the edge of the simulator and detect very few enemies. As a result, the population converged to a stationary solution since it was easy to produce. To correct this problem, a minimum velocity was enforced so that the guards must move and navigate to obtain a high fitness score.

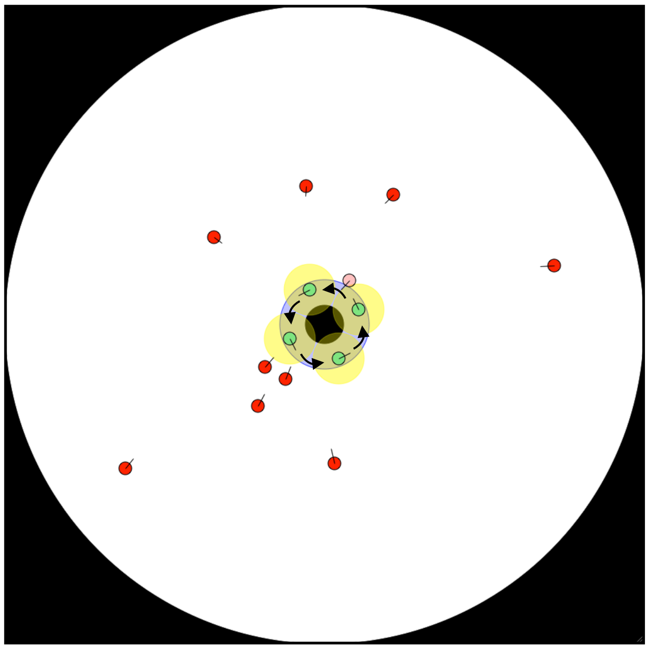

Figure 10. Continuous Result for Homogenous Teams

Figure 10 shows the solution evolved for homogenous teams in the continuous simulator. The guards immediately collapse inward towards the base and rapidly circle around it at a fixed radius that maximizes coverage of the perimeter. Unfortunately, navigation so close to the base and other guards requires extraordinarily precise movements, making this particular solution impractical on a physical system since sensor noise and inaccuracies in movement would result in collisions between agents.

5.3. Heterogenous Teams

Another evolutionary sequence evolved four populations of guards based on their starting positions. Figure 11 shows that each guard spins in circles near its starting position. This solution shows how difficult it is for heterogenous teams to cooperate effectively. Guards that evolve to move far from their starting position often collide with other guards, eliminating each other from the simulation and compromising the overall fitness score. Guards that evolve to keep their distance from the other guards typically detect a modest amount of enemies while removing the danger of a collision. The risk associated with pursuing a fitter control program causes the population to converge with a sub-optimal solution.

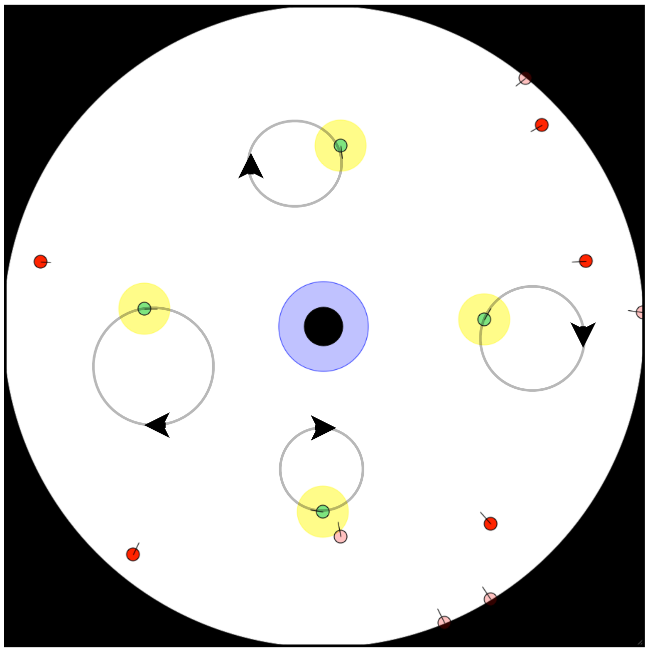
Figure 11. Continuous Result for Heterogenous Teams

5.4. Homogenous Teams with Co-Evolution

During the next evolutionary sequence enemy control programs evolved simultaneously alongside homogenous teams of guards. The primitive set and control scheme of the enemies was the same as the guards with an added terminal that returned a vector representing the position of the nearest guard relative to the enemy. The enemy's fitness score was equal to the number of enemies that reached the base's perimeter.

Figure 12 shows a solution where the guards make large circles in their respective starting regions. This control program appears to be less predictable then the solution developed by the homogenous team strategy but is much less fit when attempting to detect enemies that move straight toward the base.

5.5. Advantages of Continuous Simulation

Using a continuous simulator results in control programs that are more practical for physical agents. The distortions present in the grid-based simulator no longer affect the solutions and evolving curved navigational paths becomes possible. However, the accuracy of the continuous simulator presents a new problem. The control programs are often reckless in how close they come to other agents due to precise movements in the continuous simulator that are impractical on simple autonomous vehicles.

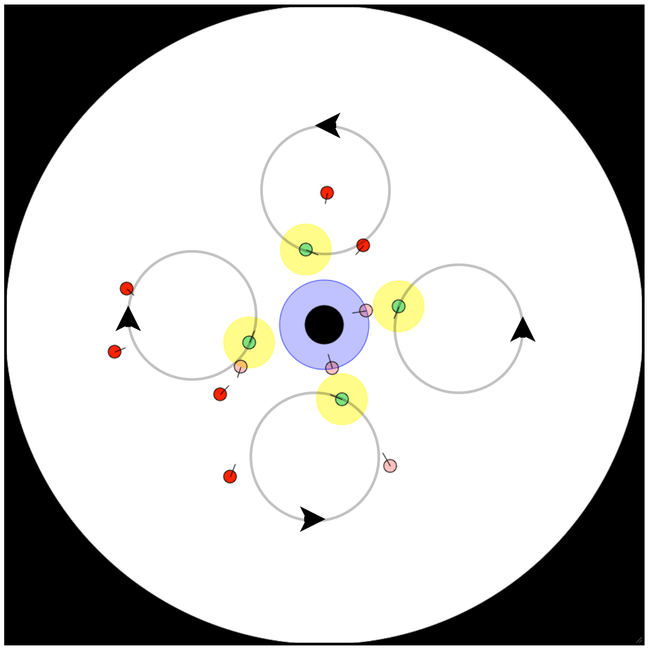
Figure 12. Continuous Result for Homogenous Co-Evolution

6. NOISY SIMULATIONS

To produce control programs that avoid reckless maneuvers, the simulator was modified by adding Gaussian noise so that movement and sensor data were no longer precise. Now, guards that perform reckless maneuvers are likely to collide with the base or each other. Therefore, simulating noisy sensors and movement produces more cautious and practical control programs.

6.1. Approximating Noise

To model sensor noise, the noisy simulator adds a vector with a uniformly random angle and a Gaussian random magnitude to the ideal sensor vector each time a control program executes the "base" or "direction" terminals. The Gaussian random magnitude has a constant variance of one unit regardless of the size of the ideal vector. For noise present in autonomous agent movement, a Gaussian random vector is also added to the ideal movement vector at each time step, however, the variance of the vector is proportional to 10% of the ideal movement vector so that faster movements are more susceptible to noise. The movement noise represents an approximation of the noise present in autonomous agents.

6.2. Primitive Set

The noisy simulator uses a primitive set that is identical to the set used for the continuous simulator, see Tables 4 and 5, with the addition of a noisy "guard" terminal that returns a vector to the nearest guard. This addition is a response to early experiments in which control programs remained near their starting positions because noise would cause them to collide with another guard when moving from their starting region. The "guard" terminal allows guards to monitor their proximity and avoid collisions.

6.3. Homogenous Teams

Homogenous teams in the noisy simulator produced the most practical autonomous agent control program. Figure 13 shows that the guards evolve to orbit the base at a reasonable distance and are able to maintain a relatively equal separation between each other. This solution is notably similar to the result of homogenous teams without noise, but is substantially more cautious in agent behavior. The amount of noise present in the system appears to determine the distance at which the guards orbit the base.

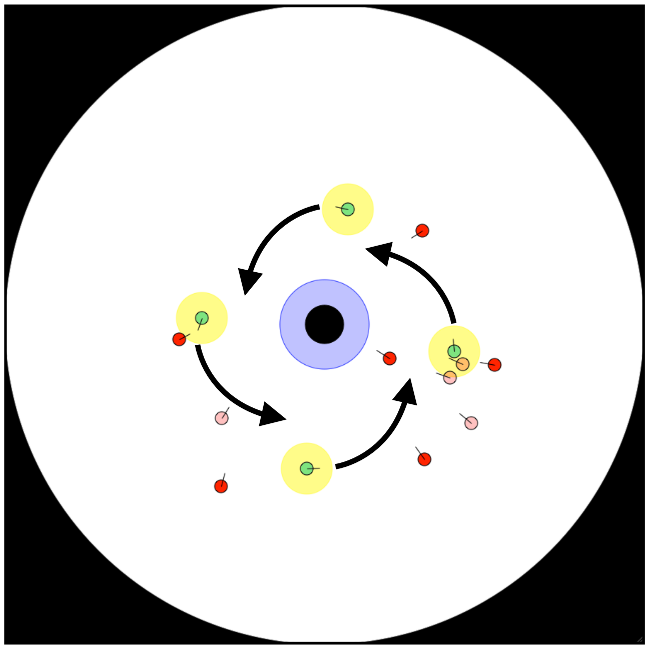

Figure 13. Noisy Continuous Result for Homogenous Teams

6.4. Homogenous Teams with Co-Evolution

An evolutionary sequence involving homogenous teams with co-evolution produced a similar result to the evolutionary sequence in the noiseless, continuous simulator. The guards navigated in much tighter circles than their noiseless counterparts, as illustrated in Figure 14. The noise once again results in similar, but more cautious control programs.

6.5. Advantages

By modeling noise in a continuous simulator, GP produces control programs that are robust enough to deal with the uncertainty noise introduces without changing the basic strategy of the guards. Guards evolved in a noisy environment were able to function in the noiseless simulator, however, the guards evolved in a noiseless environment were unable to function in the noisy simulator. These experiments illustrate the importance of evolving agents to deal with uncertainty when trying to evolve practical solutions.

7. CONCLUSION

The traditional techniques for using GP to evolve autonomous agent control programs have consequences that trivialize solutions. For example, grid-based simulations distort the space they represent, creating positions of higher value that do not exist in the real world. This research demonstrates that GP can produce control programs that are robust enough to deal with continuous and noisy environments. Similar GP techniques can be used to evolve control programs for deployment on physical autonomous agents if the simulator can accurately represent the noise present in physical sensors and actuators.

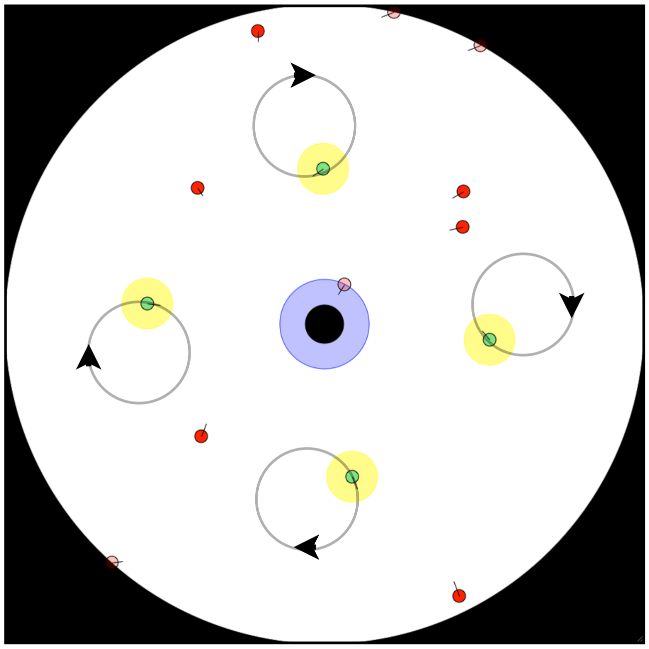

Figure 14. Noisy Continuous Result for Homogenous Co-Evolution

8. FUTURE WORK

To test the legitimacy of the techniques developed during this research, future work will focus on implementing GP evolved control programs on a physical autonomous agent. After a suitable robotic platform is selected, the characteristics of the platform must be accurately modeled in the continuous, noisy simulator. The simulator will then be used to evolve guard agents. The resulting control programs will be implemented on the autonomous agent and evaluated.

9. ACKNOWLEDGMENTS

The authors thank Dr. Arnold Patton of the Bradley University Computer Science Department for his advice and consultation throughout our research.

10. REFERENCES

[1] Koza, J. *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[2] Holland, J. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[3] Cohn, J., Weaver, J., and Redfield, S. Cooperative Autonomous Robotic Perimeter Maintenance. In *Florida Conference on Recent Advances in Robotics 2009 Proceedings* (Melbourne, Florida, May 21-22, 2009).

[4] Naeini, A. and Ghaziasgar, M. Improving Coordination via Emergent Communication in Cooperative Multiagent Systems: A Genetic Network Programming Approach. In *IEEE International Conference of Systems, Man, and Cybernetics* (San Antonio, TX, October 11-14, 2009).

[5] Luke, S. Genetic Programming Produced Competitive Soccer Softbot Teams for RoboCup97. In *Genetic Programming 1998: Proceedings of the Third Annual Conference* (Madison, WI, July 22-25, 1998).

[6] Floreano, D. and Keller, L. Methods for Artificial Evolution of Truly Cooperative Robots. In *Bio-Inspired Systems: Computational and Ambient Intelligence* (Salamance, Spain, June 10-12, 2009). Springer Berlin, Heidelberg, Germany, 2009, 15786–15790.

[7] Poli, R., Langdon, W., and McPhee, N. *A Field Guide to Genetic Programming.* Creative Commons, San Francisco, CA, 2008.