Abstract

Barlow, Gregory John. Design of Autonomous Navigation Controllers for Unmanned Aerial Vehicles Using Multi-objective Genetic Programming. (under the direction of Edward Grant.)

Unmanned aerial vehicles (UAVs) have become increasingly popular for many applications, including search and rescue, surveillance, and electronic warfare, but almost all UAVs are controlled remotely by humans. Methods of control must be developed before UAVs can become truly autonomous. While the field of evolutionary robotics (ER) has made strides in using evolutionary computation (EC) to develop controllers for wheeled mobile robots, little attention has been paid to applying EC to UAV control. EC is an attractive method for developing UAV controllers because it allows the human designer to specify the set of high level goals that are to be solved by artificial evolution. In this research, autonomous navigation controllers were developed using multi-objective genetic programming (GP) for fixed wing UAV applications. Four behavioral fitness functions were derived from flight simulations. Multi-objective GP used these fitness functions to evolve controllers that were able to locate an electromagnetic energy source, to navigate the UAV to that source efficiently using on-board sensor measurements, and to circle around the emitter. Controllers were evolved in simulation. To narrow the gap between simulated and real controllers, the simulation environment employed noisy radar signals and a sensor model with realistic inaccuracies. All computations were performed on a 92-processor Beowulf cluster parallel computer. To gauge the success of evolution, baseline fitness values for a successful controller were established by selecting values for a minimally successful controller. Two sets of experiments were performed, the first evolving controllers directly from random initial populations, the second using incremental evolution. In each set of experiments, autonomous navigation controllers were evolved for a variety of radar types. Both the direct evolution and incremental evolution experiments were able to evolve controllers that performed acceptably. However, incremental evolution vastly increased the success rate of

incremental evolution over direct evolution. The final incremental evolution experiment on the most complex radar investigated in this research evolved controllers that were able to handle all of the radar types. Evolved UAV controllers were successfully transferred to a wheeled mobile robot. An acoustic array on-board the mobile robot replaced the radar sensor, and a speaker emitting a tone was used as the target. Using the evolved navigation controllers, the mobile robot moved to the speaker and circled around it. Future research will include testing the best evolved controllers by using them to fly real UAVs.

DESIGN OF AUTONOMOUS NAVIGATION CONTROLLERS FOR UNMANNED AERIAL VEHICLES USING MULTI-OBJECTIVE GENETIC PROGRAMMING

BY

GREGORY J. BARLOW

A THESIS SUBMITTED TO THE GRADUATE FACULTY OF
NORTH CAROLINA STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

ELECTRICAL AND COMPUTER ENGINEERING

RALEIGH

MARCH 2004

APPROVED BY:

Edward Grant

CHAIR OF ADVISORY COMMITTEE

Dedicated to

my parents, John Black Barlow, Jr. and Cheryl Stevens Barlow.

Biography

Gregory John Barlow was born June 14, 1980 in Greensboro, North Carolina to John Black Barlow, Jr. and Cheryl Stevens Barlow. He graduated from the North Carolina School of Science and Mathematics, Durham, North Carolina in May 1999. He received Bachelor of Science degrees in Electrical Engineering and Computer Engineering from North Carolina State University, Raleigh, North Carolina in May 2003. He received the Master of Science Degree in Electrical Engineering from North Carolina State University, Raleigh, North Carolina in May 2004.

As an undergraduate, Gregory was a Barry M. Goldwater Scholar, a John T. Caldwell Scholar, a Caldwell Fellow, and a University Scholar. He received a North Carolina State University Undergraduate Research Award for the spring of 2001, a National Science Foundation Summer Undergraduate Fellowship in Sensor Technologies at the University of Pennsylvania for the summer of 2001, and was a winner of the 2001 North Carolina State University Undergraduate Research Symposium and the 2001 Sigma Xi Student Research Symposium.

Gregory is a member of Eta Kappa Nu Electrical and Computer Engineering Honor Society, Tau Beta Pi Engineering Honor Society, the Honor Society of Phi Kappa Phi, the Institute of Electrical and Electronics Engineers, and the International Society for Genetic and Evolutionary Computation.

Acknowledgments

I would like to thank the members of my committee, Dr. Edward Grant, Dr. Choong K. Oh, Dr. Mark W. White, and Dr. H. Troy Nagle. I would like to thank Dr. Edward Grant for all of his support and encouragement. In all the years I've been privileged to work with him, he has given me so many opportunities to learn and grow. Dr. Grant has helped me to become a better researcher and a better person. I would also like to especially thank Dr. Choong Oh for giving me the opportunity to become involved in this research. Dr. Oh has been a wonderful mentor.

I would like to acknowledge the financial support of this work provided by the Office of Naval Research (ONR) through the United States Naval Research Laboratory (NRL) under Dr. Maribel Soto (ONR) and Dr. Choong Oh (NRL). Computer time on the 92 processor Beowulf cluster was furnished by NRL (Code 5732).

I would like to thank all the members of the Center of Robotics and Intelligent Machines (CRIM), past and present, for their collaboration and support. I have spent six wonderful years in the CRIM and have had the chance to work with many great people. I would especially like to thank Andrew Nelson, John Galeotti, Leonardo Mattos, and Marc Edwards.

Most of all, I would like to thank my parents, John and Cheryl Barlow. Their persistent and wholehearted commitment to my education and growth as a person have made me what I am today. I am forever grateful. I would also like to thank my three sisters, Logan, Lindsey, and Gwendolyn, Liddy Gerchman, and all of my family and friends.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

AL Artificial Life

ANN Artificial Neural Networks

AoA Angle of Arrival

EA Evolutionary Algorithm

EC Evolutionary Computation

ER Evolutionary Robotics

EW Early Warning (Radar)

GA Genetic Algorithm

GP Genetic Programming

GPS Global positioning system

nmi Nautical Miles

RC Remote Control

RCS Radar Cross Section

TA Target Acquisition (Radar)

TT Target Tracking (Radar)

UAV Unmanned Aerial Vehicle

Chapter 1

Introduction

1.1 Motivation and Research Goals

Unmanned aerial vehicles (UAVs) have become increasingly popular for many applications, including search and rescue, surveillance, and electronic warfare, but almost all UAVs are controlled remotely by humans. For UAVs to become truly useful, especially for military applications, methods of autonomous control must be developed. While the field of evolutionary robotics (ER) has made strides in using artificial evolution to develop controllers for wheeled mobile robots, little attention has been paid to controlling UAVs. Evolutionary computation (EC) is an attractive method for developing UAV control because it allows the human designer to specify a set of high level goals to be solved by artificial evolution.

This research presents an approach to designing behavioral navigation controllers for fixed wing UAVs. Multi-objective genetic programming (GP) was used to evolve these controllers in simulation. The goal of the research was to produce controllers that could locate an electromagnetic energy source – a radar in this research – navigate the UAV to the source efficiently, and then circle closely around the emitter. Four fitness functions, based on observations of

UAV flight tests, were established to measure these behaviors: the average normalized distance to the radar, the circling distance, the amount of time spent flying with a roll angle of 0°, and a measure of the cost of turning sharply. These fitness functions were combined using multi-objective optimization. Research began by examining a very simple radar type that was continuously emitting and stationary. Experiments progressed to more difficult radar types, including radars that emitted intermittently and radars that moved around the simulation area. A second set of experiments used incremental evolution to improve the chance of evolving successful controllers.

While there has been some success in evolving controllers directly on real robots, simulation is much more common. For UAVs, simulation is the only feasible method for evolving controllers. A UAV cannot be operated for long enough to evolve a sufficiently competent controller, the use of an unfit controller could result in damage to the aircraft, and flight tests are very expensive. The development of the simulation environment is extremely important to the future success of evolved controllers. After being evolved in simulation, controllers must be transfered to real UAVs, and ensuring that the evolved controllers are sufficiently robust was an important consideration of the evolutionary process. Methods for effective transference were incorporated into the simulation by abstracting navigation control from UAV flight, tuning simulation parameters for equivalence to real aircraft, and adding noise to the simulation.

1.2 Overview of Thesis Chapters

Chapter 2 reviews literature related to the research, including the areas of evolutionary computation, genetic programming, and evolutionary robotics. Methods for measuring fitness in evolutionary robotics are also examined. Issues within the literature of particular concern to this research, including transference and incremental evolution, are identified.

Chapter 3 describes the problem of controlling UAVs to locate radars. Considerations and current trends in UAV technology are examined. The simulation environment, including models for UAVs, radar sites, and sensors, is presented. The difficulty of the problem and the issue of transference are discussed.

Chapter 4 presents the multi-objective GP algorithm used to evolve controllers. Four fitness measures were designed to promote the evolution of good behaviors: normalized distance, circling distance, level time, and turn cost. The fitness functions and strategies used to evolve controllers are presented along with the parameters used by GP. Methods of incremental evolution used to evolve controllers are also presented.

Chapter 5 describes the use of multi-objective GP to evolve controllers for a variety of radar types. First, the effectiveness of the four fitness functions in producing controllers that satisfy the mission goals is analyzed. After presenting a post-evolution metric for performance evaluation, experimental results for controllers evolved on a variety of radar types are presented. Results are presented for experiments using both direct and incremental evolution. Finally, the transference of evolved controllers to real robots is described.

Chapter 6 presents concluding remarks on the success of the research. An overview of future research is also included.

Chapter 2

Literature Review

To be truly helpful to humans, robots must be intelligent and able to function autonomously. To be considered intelligent, a robot must not perform tasks mindlessly, and to be considered truly autonomous, a robot must be able to function in a complex and noisy environment without human aid [55]. However, most of the robots known to the general public, including Honda's ASIMO humanoid robot, robots shown on television, and most industrial robots, lack both autonomy and intelligence. Instead, they are either directly controlled by humans or preprogrammed, often operating in a simplified environment. Robots have the potential to improve our lives in many ways, performing difficult, dangerous, and repetitive tasks, but to make this truly possible, robots must be intelligent and autonomous.

Since neurophysiologist W. Grey Walter designed *Machina speculatrix* in the early 1950's [77], the area of autonomous robots has been an active area of research. The *Machina speculatrix*, a robot which Walter also called a tortoise, was a three-wheeled robot with a brain composed of only a few analog neurons. The tortoise displayed simple behaviors including returning to a recharging station when its battery was low and homing on a light source. Walter showed how complex behavior could be obtained from a simple controller able to interact with a physical environment.

As the field of artificial intelligence (AI) grew, so did research on autonomous robotics. The dominant controller architecture used for autonomous agents was the classical [70] or hierarchical [55] architecture. In this architecture, the robot senses the world, plans the next action, and then acts. An internal global world model is maintained to be used by the planner. The planning stage might take a long time, and changes in the environment might be ignored. The earliest example of a robot using this architecture is Shakey [61], a robot designed at the Stanford Research Institute. Shakey used the STRIPS algorithm to generate plans.

In the mid-1980's, Brooks suggested an alternative to the classical architecture [7], which he would later call the subsumption architecture [8]. In the subsumption architecture, sensing and action are tightly linked, and the world is used as its own model [9]. This research has expanded into the field of behavior-based robotics [3]. Behavior-based robotics differs from the classical architecture in the diminished importance of planning. While some behavioral architectures still use planning, there is a much tighter reactive linking between sensing and action than in the hierarchical architecture [55]. Behavioral architectures that are completely reactive, with no planning at all, are called "model-free" controllers. Model-free controllers have no internal environmental model, and effector outputs are a direct function of sensor inputs [35].

2.1 Evolutionary Robotics

The field of evolutionary robotics (ER) [62] combines research on evolutionary computation (EC) [5,31] with behavior-based robotics and artificial life (AL) [1]. The primary goal of ER is the automatic design of behavioral controllers with no internal environmental model. ER uses a population-based evolutionary algorithm to evolve autonomous robot controllers for a target task.

Many evolved robot controllers are model-free. It is possible to include explicit environmental models for use by evolved controllers [35], but most research does not. While it is feasible for evolution to produce internal world models, most environments investigated in ER are not sufficiently inaccessible to require it. The evolution of internal models is also unlikely using current ER techniques.

2.1.1 History of Evolutionary Robotics Research

Some of the earliest experimental ER work was done by Koza [40]. He used genetic programming (GP) [39] to evolve a subsumption controller for the wall-following problem approached by Mataric in [52]. The program evolved by Koza was similar in size to the subsumption programs written by Mataric. While the program evolved by GP was intended to control a real robot, tests were done only in simulation, making it difficult to gauge the true effectiveness of the evolved controller.

The research group at the University of Sussex [12] applied the term "evolutionary robotics" to this field in 1993. Much of the early ER work was done by the Sussex group [12, 28, 29, 33] or the collaboration between Stephano Nolfi, Dario Floreano, and others [21, 50, 62, 64]. Reviews of the field of ER at various stages can be found in [29, 51, 57, 62].

Much of the early ER research studied obstacle avoidance navigation, and the evolution of this type of behaviors continues in current research. Nolfi, Floreano, Miglino, and Mondada [64] evolved navigation and obstacle avoidance on a real robot that used 8 infrared (IR) sensors and was controlled by an artificial neural network (ANN). Successful behaviors were produced after 100 generations, which took about 60 hours, since evolution was embodied on actual robots. Jakobi, Husbands, and Harvey [33] evolved obstacle-avoiding behaviors in a simulated robot controlled by an ANN and then successfully tested the evolved controller on a real robot.

Edner [16] evolved a GP controller for obstacle avoidance, first in simulation and then on a real robot. Filliat, Kodjabachian, and Meyer [19] evolved obstacle avoidance in a six-legged walking robot. In their research, the ANN controller was evolved in simulation, and then the final controller was successfully tested on a real robot.

Homing is another common behavior studied in ER research. Floreano and Mondada [21] evolved a neural network controller that could locate and periodically return to a battery charger when energy levels got low. Evolution took place entirely on real robots. Lund and Hallam [47] evolved a controller for a similar task, but evolution took place in simulation. They tested the evolved ANN on a real robot, with behavior similar to simulation.

Another problem often studied in ER research is object movement. Lee, Hallam, and Lund [43] evolved a GP controller to solve a box-pushing problem. The controller was evolved in simulation and then transferred to real robots. Results showed that the evolved controller was very capable at the task and there was no loss of performance when the controller was transferred to a real robot. Kamio, Misuhashi, and Iba [34] evolved a GP controller for a walking robot to push a box to a goal. The controller was evolved in simulation, and then transferred to a real robot, where the controller was refined using Q-learning. Liu and Iba [44] evolved GP controllers for two robots, one serving as a “hand”, the other as an “eye”. The controller was evolved in simulation to carry an object to a goal. The controllers have been transferred to real robots, but results on the success of transference are not yet available.

Another class of problems that has been studied in ER is game playing. Evolving robots to play games can be more difficult than evolving simple behaviors, since successful strategies for playing games may require multiple behaviors. Hsu and Gustafson [32] evolved agents to play keep-away soccer. Their GP controllers were not tested on real robots. Nelson et al. [57–60] evolved ANN controllers to play capture the flag. Evolved controllers were competitive with rule-based controllers, and transferred well to real robots.

In many of these experiments, the problems to be solved were designed specifically for research purposes. While simple problems generally require a small number of behaviors, more complex real-world problems might require the coordination of multiple behaviors in order to achieve the goals of the problem. Very little of the ER work to date has been intended for use in real-life applications.

2.1.2 Evolutionary Robotics Controller Architectures

A variety of controller architectures have been used in ER. The subsumption architecture was used by Koza [40] early in ER research, and continues to be used today by some researchers [44]. Genetic algorithms have been used to evolve rule sets for control [49, 79]. The use of evolvable hardware for robot control has also been studied [35]. The two most popular control architectures, however, are artificial neural networks and genetic programming.

ANNs consist of many simple processing units, or neurons, with many interconnections [20]. ANNs are the most popular controller structure used in ER research [12, 19, 21, 28, 29, 33, 47, 50, 57–60, 62, 64]. To speed up evolution, many studies restrict the complexity of network topologies [21]. In some studies, the network topology is allowed to evolve along with the weights [45, 54, 57], vastly increasing the search space, but making the use of ANNs more scalable to difficult problems.

Despite Koza's early ER work with GP, ANNs have dominated ER research. However, GP has also been shown to produce functional behaviors for autonomous robot control [16, 32, 34, 43, 44, 65]. While ANNs only require the designer to specify constraints on the network topology, GP requires additional human involvement prior to evolution. The designer must specify the functions available to evolution, a process which does introduce human bias into the evolutionary process. For real-world problems, however, constraining evolution by using

GP is often not such a disadvantage; the natural constraints of the problem often make function selection more obvious.

2.1.3 Simulation

Early in ER research, Brooks noted that the evolution of robot controllers would probably need to occur in simulation [10]. While some controllers have been evolved *in situ* on physical robots, evolution requires many evaluations to produce good behaviors, which generally takes an excessive amount of time on real robots. Evolving controllers in simulation is less constraining, because evaluations are much faster and can be parallelized. Since simulation environments cannot be perfectly equivalent to the conditions a real robot would face, transference of controllers evolved in simulation to real robots has been an important issue.

Early in ER research, many studies evolved controllers directly on physical robots [21, 62, 64]. Though the availability of computational power has made simulation increasingly attractive, some research using embodied evolution continues [16, 50]. Embodied evolution is often preferable to evolution using simulation because a model of the world doesn't need to be created for simulation. Embodied evolution can directly test the performance of controllers on the exact problems one is interested in solving, including the noise from the actual environment. However, embodied evolution is very slow, which constrains the complexity of problems that can be solved in a reasonable amount of time.

Simulation has been used since the beginning of ER research [40] with varying success for the control of real robots. Some simulated controllers are never tested on actual robots, and some fail to transfer well. However, many controllers evolved in simulation have been successfully transferred to real robots [19, 29, 33, 34, 43, 47, 57, 59, 60]. Adding noise to the simulation is one method that has proved successful in evolving controllers that transfer well to real robots. This technique was introduced in [33] by Jakobi et al. with good success.

2.1.4 Robot Types

A variety of robot types have been used in ER research. By far the most common are wheeled mobile robot experiments [16, 21, 29, 33, 35, 43, 44, 47, 50, 57–60, 62, 64]. The most popular wheeled robot in use for ER research has been the Khepera [21,29,33,43,44,47,62,64]. Some researchers using the Khepera have now begun using a larger robot by the same manufacturer called the Koala [50]. Other research has used existing wheeled research robots [16, 35]. At North Carolina State University, the EvBot [24] and EvBot II [53] mobile robots have been designed specifically for ER research [57–60].

ER research has extended to other robot types as well. Some researchers have done ER experiments with walking robots. Filliat et al. [19] evolved locomotion and obstacle avoidance for a walking robot with six legs. Kamio et al. [34] evolved a controller for a Sony AIBO four-legged walking robot. The goal of this robot was to solve a box pushing task. At the University of Sussex, Harvey, Husbands, Cliff, Thompson, and Jakobi [28, 29] used a specialized gantry robot to do ER research. This robot was composed of a camera assembly suspended from a gantry. The camera was aimed at a mirror angled at 45° so the camera could see in a way similar to a camera mounted on a wheeled mobile robot. The mirror could be rotated to change the "direction" this robot was facing. The gantry enabled the camera mechanism to be moved within a plane. This gantry robot could be used much like a wheeled mobile robot. A variety of ER experiments including vision work were done with this robot.

A robot type that has received relatively little attention has been the unmanned aerial vehicle (UAV). ER experiments on flying vehicles can be separated into two general classes of aircraft: fixed wing and rotary wing. The most common type, especially for military applications, is the fixed wing UAV [14]. Hoffmann, Koo, and Shakernia [30] evolved a rotary wing helicopter autopilot using evolutionary strategies to evolve a fuzzy logic rule base. Shim, Koo, Hoffmann, and Sastry [72] compared this approach to linear robust multi-variable control and nonlinear tracking control in simulation. They showed that the evolved controller was able to handle uncertainties and disturbances. Marin, Radtke, Innis, Barr, and Schultz [49] evolved a controller for a UAV of an unspecified type. They evolved a set of rules to reactively control a UAV's flight based on target detection. Their experiments were only in simulation, the movement of the UAV was grid-based, and the UAV could move in any direction at every time step. Because of the unrealistic nature of the simulation, it would have be difficult to control real UAVs with the evolved controllers. In related work, Wu, Schultz, and Agah [79] evolved a control scheme for micro air vehicles. Their evolved control system was distributed. Each vehicle had its own controller, though all controllers were identical. Rule sets were evolved to control the UAVs. Like the experiments in [49], only simulation was used, simulation was unrealistic, and no testing on real UAVs was attempted. Meyer, Doncieux, Filliat, and Guillot [54] evolved a neural network control system for a simulated blimp. Unlike rotary wing and fixed wing UAVs, a blimp is very stable and easy to pilot. The goal of the research was to develop controllers capable of countering wind to maintain a constant flying speed. The evolved control system was only tested in simulation.

2.2 Genetic Programming

Genetic programming (GP) is a subfield of evolutionary computation (EC) [5] that uses a genetic algorithm (GA) to evolve computer programs. EC uses a computer model of the naturally occurring evolutionary process to solve problems [31]. A GA, a type of EC algorithm, operates on a population of individual objects, each with its own fitness, using genetic operators like recombination and mutation to create a new generation. As this process is repeated, the principle of survival of the fittest improves the fitness of the population.

GP uses a GA to evolve populations of computer programs. The GP paradigm was introduced and championed by John Koza [36]. Koza has shown that GP is not simply random search, it is much more effective [39]. GP allows for individuals to vary in size, which is necessary for computer programs of varying lengths. Like most subfields of EC, GP is domain-independent. GP has been applied to a diverse selection of problems. Extensive information on GP is available in Koza's books [36–38].

2.2.1 Evolutionary Process

GP represents an individual as a tree, where each node represents a programming command. Each non-leaf node is a function, which takes at least one argument, and each leaf node is a terminal, which takes no arguments. A GP program tree can also be seen as an S-expression similar to those in LISP, though GP systems are written in many languages.

The most common method for evolving computer programs using GP is generational evolution. First, an initial population of random computer programs is generated. This population can be generated in several ways, including setting the initial depth of all individuals to be the same, ramping the initial depth of individuals over the population, or mixing these methods. After the initial population has been created, evolution begins. During each generation, GP evaluates each individual in the population and assigns it a fitness value using a fitness function. Then, a new generation is formed by probabilistically selecting one or more parent programs from the population and applying genetic operators to them. Three genetic operators are typically used, and the percentage of time each is applied is set as a parameter of evolution. Reproduction directly copies a selected individual to the new population. Crossover recombines two parent programs to create new children. Mutation creates one new individual by mutating a single parent. Mutation is typically done by selecting a node within the tree, destroying the subtree below that point, and creating a new random subtree in its place. The mutation rate, the percentage of the time that mutation is applied, is usually set to be very small for GP, as mutation is very disruptive. This operator is intended to aid in broad search and to help keep GP from being stuck in local minima. Evolution takes place over a specified number of generations or until a success criterion is satisfied. Koza [36–38] provides extensive information on the specific implementation of GP in software.

2.2.2 Concerns and Strategies

While the general framework of GP is well defined, many parameters of the algorithm can be altered. There have been large numbers of studies investigating all aspects of GP in the hope of making the evolutionary process efficient and successful. Two important parameters for controlling GP are the population size and the maximum number of generations. Crossover benefits from a large population, though too large a population can make propagation of good behaviors difficult. Another parameter for controlling evolution is the ratio between crossover and mutation operations. Typically, GP uses the crossover operation more than 90 percent of the time. Evolution can also be controlled by varying the crossover and selection types [38]. The evaluation method is also important to the success of GP. Panait and Luke [67] investigated the impact of sampling method on robustness. They found that the best sampling method was domain-dependent, demonstrating the need to take GP parameters into consideration before beginning evolution.

An important consideration in GP research has been code growth, or bloat. Programs evolved with GP tend to grow over time without necessarily increasing in fitness [75]. This is a problem not only because code growth makes it difficult to tell what the program is actually doing, but bloat also hinders the progress of evolution, creating stagnation [6]. Bloat tends to be made up of neutral code, or introns, which are branches of the program tree which are never executed under any circumstance or have no effect on the program's outcome. Several studies of the causes of bloat have been pursued [6, 75]. These works studied several possible causes of bloat, but the main conclusion was that bloat is problem independent. A variety of techniques have been devised to contain code bloat. Perhaps the most popular has been a fixed limit to the depth of programs. Generally, this limit is constant over all generations, though Silva and Almeida [73] have proposed a dynamic depth limit. Luke and Panait [46] have proposed a method known as lexicographic parsimony pressure where size is taken into account during selection. This technique can be used in combination with depth limiting.

2.3 Performance Evaluation

One of the great challenges of EC is the formulation of fitness functions for performance evaluation [5]. A fitness function (alternatively called a fitness measure, fitness metric, or objective function) allows an individual within a population to be ranked based on its performance. This ranking is used for selection in the evolutionary process. The success of evolution is heavily dependent on the formulation of an effective measure of fitness.

For many applications, EC is used for the purpose of global optimization. In ER, the evolutionary process is used instead for primary generation. For a sufficiently complex task, finding the absolute optimal response is generally not of interest. Instead, the goal is to generate a desired set of complex behaviors or to successfully complete a desired task.

In a reactive controller, the robot receives sensor signals and then generates actuator commands over many time steps. As the robot moves, its relationship to the environment changes, changing the sensor values. The necessary actuation values to produce successful behaviors are rarely known *a priori*; if they were, evolution would be unnecessary. Because the actuation values for a desired behavior are generally not known, the fitness of actuator outputs cannot be directly measured. Instead, populations of robot controllers must be evolved using fitness functions that measure the expression of behaviors, rather than a sensor-actuator input-output error.

In [62], Nolfi and Floreano describe a *fitness space* to classify fitness functions for autonomous systems. Their fitness space has three continuous dimensions: functional-behavioral, explicit-implicit, and external-internal. A given fitness function can be mapped to a point in this three-dimensional space. The *functional-behavioral* dimension indicates what the fitness function measures, functioning modes or behavior of the controller. For evolving robot behaviors, a functional fitness metric would measure a sensor-actuator input-output error, while a behavioral metric would measure the expression of desired behaviors. The *explicit-implicit* dimension is defined by the number of variables and constraints in the fitness function. Implicit functions tend to measure only very high level behaviors and have very few variables. Explicit functions measure more subsystems and have more variables. The *external-internal* dimension indicates whether the variables and constraints used to compute the fitness are directly available to the evolving robot. External functions use require information not available to the robot; internal functions require only information on-board the robot. The best location for a fitness function in this three-dimensional space depends on the problem. For the evolution of autonomous systems, a fitness function that leans toward the behavioral, implicit, internal corner of the fitness space is best. If evolution is done in simulation, the *external-internal* dimension is mostly unimportant.

While fitness function selection is perhaps the most challenging part of applying ER to complex, real-world problems, there are a number of open issues which require further resolution. First, there is a need to select for fitness in initial populations that have little to no measurable ability to complete the overall tasks, which might be very complex. If the fitness measures are so difficult that initial populations are comprised entirely of individuals with no detectable level of fitness, then the evolution cannot effectively select more fit individuals. This is commonly called the Bootstrap Problem, and can lead to populations that stagnate easily. Second, there is a desire to minimize human bias in the production of fitness measures. As problems become more complex, effective fitness functions can no longer be designed based on human knowledge of the domain. Third, evolution should be able to generate behaviors for multiple objectives which might conflict. ER generally uses only a single fitness function, allowing individuals in the population to be ranked very easily. For problems with only a single objective, this poses no difficulties. When a problem has multiple objectives, fitness function design and evolution's method of selection must be altered. Fourth, ER should be able to evolve controllers that are capable of controlling real robots in realistic, real-world environments. All too often, ER research has been done in simple, unrealistic domains, and has been completely unsuitable for real-world applications.

Researchers have adapted a variety of strategies to address these problems. At the present time, no single method has shown itself to be preferable for all problems. In this section, several of the more successful fitness function strategies are presented and discussed. While some of these methods are mutually exclusive, some are complementary.

2.3.1 Functional Fitness Functions

Functional fitness functions are a quantification of an individual's fitness measuring some aspect of performance beyond a binary success measure. Functional fitness functions are often formulated by trial and error or a human designer's expertise (or both). Using functional fitness functions introduces human bias into fitness function design because the human designer must decide how important a particular behavior is to the overall fitness of the individual.

Traditional functional fitness functions, which produce a single value for each individual in the population, are most useful for tasks that have a single, measurable objective. However, functional fitness functions can be used to optimize over multiple objectives. To accomplish this, a fitness function for each objective is formulated, and then a single function for all the objectives is produced by summing each objective's function. Typically, these functions are weighted to give precedence to one or more objectives, and this weighting may introduce a great deal of human bias into fitness function design.

Many ER experiments have used functional fitness functions. In [28], Harvey et al. designed fitness functions to measure the distances between a robot and a series of targets where the task was for the robot to move toward the target. Floreano and Mondada [21] used a functional fitness measure to evolve a robot to navigate and avoid objects. The fitness function was designed to maximize speed, straight line motion, and obstacle avoidance. These three components were combined into a single function. Lee et al. [43] designed three fitness functions, the first for simple box-pushing, the second for box-side-circling, and the third to combine the previous two behaviors into a high level behavior. The first two fitness functions combined multiple objectives into single value functions, while the third fitness function measured only a single value.

2.3.2 Aggregate Fitness Functions

In a very different manner from functional fitness functions, aggregate fitness functions measure only the high-level success of individual controllers. Aggregate fitness selection (or "all-in-one" evaluation) measures the fitness of an evaluation trial as a single binary value. The final fitness is the sum of values from a number of trials.

While functional fitness functions often require a great deal of human bias, aggregate fitness functions require much less human domain knowledge, reducing the amount of human bias in the fitness function design. This reduction of human bias is viewed by some as extremely necessary for the evolution of truly complex behaviors [57]. Because aggregate fitness functions judge a particular trial as either a success or a failure, they are particularly useful in competitive evolution, described in Section 2.3.3.

Despite these advantages, aggregate fitness functions also have many disadvantages. This method has never been very popular in ER research because it is extremely subject to the Bootstrap Problem (described in Section 2.3), where initial populations have no detectable fitness. One method for overcoming this problem, a bimodal fitness function, was examined in [57]. Aggregate fitness functions also require a task where success can be clearly defined. Since fitness is all or nothing, individuals that come very close to fulfilling a task receive the same score as individuals that fail completely. Likewise, there is little evolutionary pressure for an individual to improve beyond simply satisfying the goal. Also, if the problem in question does not have an implicit boundary between success and failure, human bias must be introduced, negating one of the major advantages of this technique.

2.3.3 Competitive Fitness Evaluation

Many researchers have examined competitive fitness evaluation as a method for overcoming some of the current challenges related to fitness functions design in ER. Competitive fitness evaluation uses direct competition between members of a population to direct evolution. Aggregate fitness functions are typically used to evaluate the results of competition, as mentioned in Section 2.3.2. Because individuals compete head to head with one another, the performance of an individual is directly affected by the fitness of other individuals in the population. The attraction of competitive fitness evaluation is the continual evolutionary pressure co-evolving populations exert on each other. As one population evolves more fit behaviors, the evolutionary pressure on the other increases, creating increasingly more complex behaviors.

Co-evolution can be implemented in two ways: with a set of multiple co-evolving populations, or with competitive selection in a single population. Examples in the literature [45, 63] demonstrate the ability of co-evolving populations to continually change the fitness landscape, promoting increasingly competent individuals. This trend of reciprocal increases in difficulty is also seen in co-evolution using only a single population. As the fitness of individuals in the population increases, the fitness landscape is altered, and the problem becomes more difficult. This is known as the "Red Queen Effect" [22] for Lewis Carroll's character in *Through the Looking Glass* whose running never yielded any advancement, because the landscape moved with her. There are successful examples of single population co-evolution in the literature that demonstrate this effect [57, 67]. Competitive fitness evaluation allows the dynamics of evolution to exert pressure on populations to develop increasingly competent individuals. In more traditional EC work, once the population has satisfied the objectives in the fitness function, evolution has no incentive to continue to get better. This is a major advantage of co-evolution.

Despite the advantages of competitive fitness selection, this method is not perfect. First, for multiple populations, the level of play possible depends highly on the opponent. If the competition is uneven, and one population has a more difficult task, or if one population is significantly better than the other, evolution can stagnate. Also, co-evolution works best for problems where there is already competition, like game playing [45, 57] or predator-prey problems [22, 63]. Without inherent competition, co-evolution usually requires changing the problem to fit the competitive fitness model. It is possible to use co-evolution more generally by using two populations; one population contains regular structures to be evolved, the other contains training cases. Using co-evolution can help concentrate fitness calculations on difficult training cases. The results in [67] suggest that co-evolution used in this manner is still very domain-dependent. For some tasks, competitive fitness evaluation should be considered an excellent option, but it cannot be applied equally well in all cases.

2.3.4 Multi-objective Optimization

Many problems have multiple objectives, and in order to optimize over more than one objective, EC techniques must be enhanced beyond a simple functional fitness function measuring only one parameter. It is possible to continue using a single-value functional fitness function by combining multiple objectives into a single function, as described in Section 2.3.1. However, this generally requires weighting, which forces the human designer to scale all functions appropriately so as to emphasize them in order of importance. While this has been successful [21, 28, 43], this method becomes more difficult as the complexity of the problem increases. The human bias in choosing weights for each component of the function plays a major role in the success or failure of evolution.

An alternative to conventional, single-valued fitness evaluation is the use of multi-objective optimization, where the evolutionary algorithm optimizes over multiple fitness measures instead of a single function. Multiple objectives may conflict with one another, so that there may not be a single optimal solution to the problem. Instead of finding a single best point based on some human weighting of the fitness function, multi-objective optimization finds a set of solutions known as the Pareto front where no individual has more optimal fitness values for all functions than any other in the front [13].

Fitness values for multiple objectives cannot be ranked as simply as those for a single objective. Multi-objective optimization uses a technique called a non-dominated sort to rank the members of the population based on their fitness values. Individuals are sorted into ranks based on their level of non-domination. The individuals in the first rank are not dominated by any other individual in the population; no individual performs worse on all fitness functions than another individual. Once the first front is found, the individuals in that front are set aside and the process is repeated with the remaining members of the population [15]. A variety of multi-objective optimization algorithms have been presented in the literature [13, 41, 42]. One of the most successful algorithms has been the *Non-Dominated Sorting Genetic Algorithm II*, or NSGA-II [15]. Multi-objective optimization has been applied to a variety of media, including GP [69].

Multi-objective optimization produces a Pareto front of solutions, which can hinder the selection of a single best solution. Typically, the resulting evolved population must be evaluated differently than it was during the evolutionary process. The decision about which member of the population represents the best solution must usually be made by the human designer. No matter the amount of human bias that contributes to this final selection, using multi-objective optimization still frees the evolutionary process from the bias that accompanies complex functional fitness functions.

2.3.5 Incremental Evolution

Incremental evolution is a method that has demonstrated success both in overcoming the bootstrap problem and in evolving complex controllers. Incremental evolution is the process of evolving a population on a simple problem and then using the evolved population as a seed to evolve a solution to a more complex problem. Several increments can be undertaken before training on the final problem.

Incremental evolution is appropriate for “hard” problems where evolution finds either the bootstrap problem or producing a successful controller difficult. In [11], Clark and Thornton classified problems into two types. Problems of *type-1* are statistical; these problems can be solved using observable statistical effects. Problems that are not *type-1* are *type-2*, problems Clark and Thornton call relational. Earlier, Elman [17] had introduced the concept of incremental learning to the training of neural networks. He studied grammar acquisition by a neural network using backpropagation, and found that training the network only on complex examples failed. However, by introducing training data incrementally - simple examples first and then more complex examples - a network could be evolved to learn the grammar acquisition task. Clark and Thornton note Elman’s results as evidence of the ability of incremental learning to solve *type-2* problems. Nolfi and Floreano [62] discuss these results in regard to evolution.

There are two types of incremental evolution. In functional incremental evolution, the difficulty of the fitness function is incremented. The purpose of this is usually to evolve a controller for a specific set of behaviors which the human designer believes are useful for accomplishing the final task. Functional incremental evolution has been used successfully by a number of researchers. Harvey et al. [28] used incremental evolution to evolve neural network robot controllers guided by vision. The networks were first evolved to move toward a large rectangular target. Networks were then evolved to find a smaller target, and then a moving target. Finally, in the portion of incremental evolution where the fitness function was significantly altered, the networks were evolved to avoid a rectangular target while moving toward a triangle. Gomez and Miikkulainen [26] evolved an enemy evasion behavior using incremental evolution and found it to be superior to direct evolution on the final problem. They first evolved a neural network to avoid a single slow enemy, then two slow enemies, and finally two fast enemies. In [27], they used a similar technique to evolve a controller for a finless rocket. Filliat et al. [19] used incremental evolution to evolve a controller for a legged robot. Locomotion was evolved first, and then obstacle avoidance. While most of the incremental evolution work has used neural networks, Winkeler and Manjunath [78] used incremental evolution with genetic programming to evolve a controller to keep a moving object in the center of a camera's field of view. Solutions evolved using incremental evolution were often superior to those evolved through direct evolution. GP has also been used in other incremental evolution experiments [32, 43]. Functional incremental evolution has met with criticism because evolution is very guided. In some cases, it is inappropriate to consider the evolved controllers as having evolved novel behaviors.

In the second method of incremental evolution, the difficulty of the individual's environment is incremented. For this to be different than simply incrementing the difficulty of the fitness function, the fitness measure generally must be behavioral. While less popular than functional incremental evolution, environmental incremental evolution has been used with success by some researchers. Harvey et al. [28] changed the fitness function as part of their incremental evolution scheme, but the first step of incremental evolution was environmental; the only change was the size of the rectangular target. Nakamura, Ishiguro, and Uchikawa [56] used environmental incremental evolution throughout the evolution of a controller that would find a peg, lift it, and remove it from the arena. The research identified a number of simple, ordered sub-behaviors that a very fit controller would need to execute to be successful: *explore→find the peg→grasp→carry toward the wall→release the peg outside the arena*. While functional incremental evolution would have required a fitness function for each stage of the task, Nakamura et al. used environmental incremental evolution. They evolved a controller in three stages: 1) the robot, already grasping the peg, is placed randomly, 2) the robot is positioned with its arm down at the front of the peg but not grasping it, and 3) the robot is positioned with its arm up at the front of the peg but not grasping it (the *explore* and *find the peg* behaviors were not evolved). The fitness function remained the same for all stages of evolution. This technique evolved a successful controller, while direct evolution on the final problem never showed appreciable fitness. Nelson [57] used environmental incremental evolution to evolve neural network robot controllers to play capture the flag. Avraham, Chechik, and Ruppin [4] used incremental evolution to evolve a neural network solution for a generalized XOR problem for a mobile agent. The sizes of objects to be recognized were made incrementally harder to enable the evolution of a successful network.

While incremental evolution has worked well in many cases, it does have some problems that must be kept in mind. First, using incremental evolution requires the human designer to play a much larger role in the evolutionary process. The selection of training increments heavily influences the success of evolution, and for many problems, these training increments are not obvious. If the wrong increments are chosen, evolution may fail, especially for functional incremental evolution, which can force the use of particular behaviors. Another consideration is the diversity of the population during incremental evolution [18]. If a population is overtrained on a simple task, there may not be sufficient diversity available for complex tasks. For mutation-only systems, like [57], this is not much of a concern, but for crossover-dependent evolution that depends on diversity, it must be kept in mind.

In this research, controllers were evolved over multiple goals using multi-objective optimization. Both function and environmental incremental evolution were also used to improve the ability of evolution to produce fit controllers for complex radar types. The simulation environment, described in Chapter 3, was designed to promote the evolution of controllers that could be transferred to real UAVs. The multi-objective GP algorithm and techniques of incremental evolution using in this work are described in Chapter 4.

Chapter 3

Unmanned Aerial Vehicle Control

The recent successes of the unmanned aerial vehicle (UAV) have helped prove it to be a cost-effective platform for military operations. The many cost and flexibility advantages of UAVs makes them attractive for many applications, including surveillance, search and rescue, and electronic warfare. At present, UAVs are usually remotely controlled by humans, but as UAV technology is deployed more widely, the need for autonomous control will grow.

The goal of this research was the development of a navigation controller for a fixed wing UAV. The UAV's mission is to autonomously locate, track, and then orbit around a radar site. There are three main goals for an evolved controller. First, it should move to the vicinity of the radar as quickly as possible. The sooner the UAV arrives in the vicinity of the radar, the sooner it can begin its primary mission, which may be jamming the radar, surveillance, or another of the many applications for this type of controller. Second, once in the vicinity of the source, the UAV should circle as closely as possible around the radar. This goal is especially important for radar jamming, where the distance from the source has a major effect on the necessary jamming power. Third, the flight path should be stable and efficient; the roll angle should change as infrequently as possible, and any change in roll angle should be small. Making frequent changes to the roll angle of the UAV could create dangerous flight dynamics and reduce the flying time and range of the UAV.

Behavioral navigation controllers for fixed wing UAVs were evolved using multi-objective genetic programming. While there has been success in evolving controllers directly on real robots [64], simulation is the only feasible way to evolve controllers for UAVs. A UAV cannot be operated continuously for long enough to evolve a sufficiently competent controller, the use of an unfit controller could result in damage to the aircraft, and flight tests are very expensive. For these reasons, the simulation must be capable of evolving controllers which transfer well to real UAVs.

In this chapter, the problem of UAV control and a description of the simulation used by evolution are presented. First, an overview of the simulation environment is given. Next, the models for UAVs, radars, and sensors used in the simulation are presented. Finally, the difficulty of this problem for both human designers and evolution is described and issues of transference to real UAVs are addressed.

3.1 Simulation Environment

The simulation environment is a square, 100 nautical miles (nmi) on each side. Two types of agents exist in this environment: UAVs and radars. The UAV model is described in Section 3.2. The radar models available in the simulation are presented in Section 3.3. In addition to UAV and radar agents, the simulation models propagation of electromagnetic energy throughout the environment. The sensor model used in the simulation is described in Section 3.4.

The fitness of a controller is calculated by running some number of simulation trials. The initial positions of the UAV and the target radar are set randomly at the beginning of each simulation.

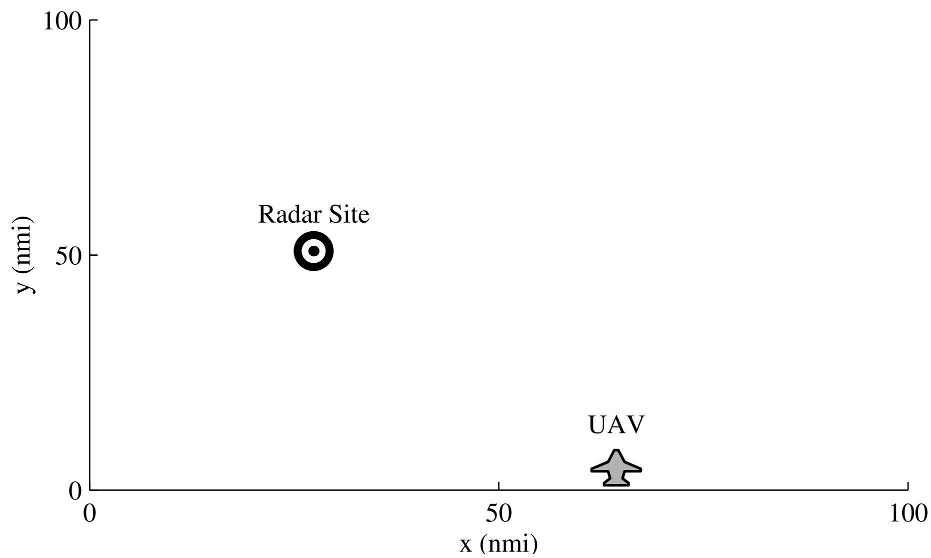


Figure 3.1: The simulation area, as shown, is 100 nmi by 100 nmi. The UAV is placed randomly along the southern edge of the simulation area, and the radar is placed randomly anywhere within the environment.

The simulator gives the UAV a random initial position in the middle half of the southern edge of the environment with an initial heading of due north and the radar site a random position within the environment every time a simulation is run. An example of these random placements is shown in Figure 3.1. Each trial involves four hours of simulated time. The simulation period is divided into time steps, each one second long. The state of the world and the state of each agent in the world update at every time step.

3.2 Unmanned Aerial Vehicles

A UAV can be viewed as an extension of several more familiar vehicles. First, a UAV can be seen as a sophisticated remote control (RC) plane. The primary differences between a UAV and an RC plane are that a UAV usually has a more complex flight control system, on-board sensors, and can be flown beyond line of sight. Second, a UAV can be seen as a flying robot,

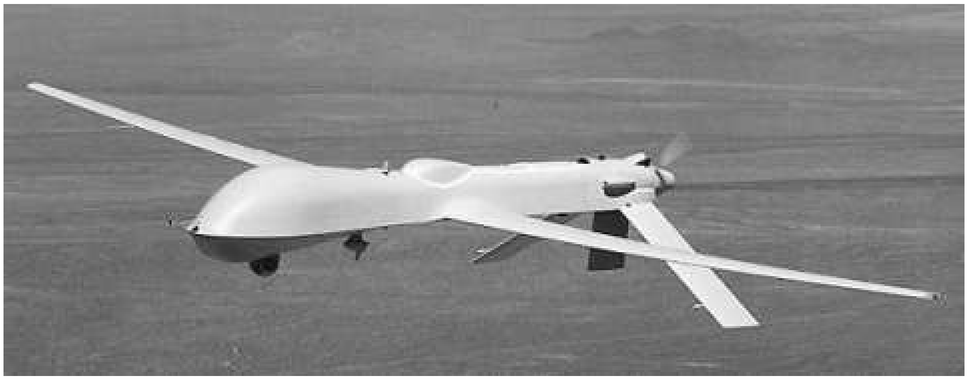


Figure 3.2: The Predator medium altitude long endurance unmanned aerial vehicle.

a view which is adopted in this work. The complex control system, on-board sensors, and potential for autonomy are all characteristics of autonomous robotic systems.

3.2.1 Characteristics

UAVs can be divided into two general classes: fixed wing and rotary wing. Fixed wing UAVs are the most common type used for military applications. The potential applications of fixed wing UAVs are similar to the tasks that planes now perform. Rotary wing UAVs, also known as helicopters, are applicable to some of the same tasks as fixed wing UAVs, though helicopters are more useful in urban areas and other areas where fixed wing aircraft have difficulty maneuvering. Like their manned counterparts, fixed wing UAVs are typically much faster than helicopter UAVs and have much greater flight range. This research focused only on fixed wing UAVs, though it might be possible to transfer these controllers to helicopter UAVs.

UAVs exist in a variety of sizes. The range and payload capacity, which depend largely on the wingspan, dictate the mission length, number of sensors, and general range of applications of a UAV. Of military UAVs in use today, the Predator UAV, shown in Figure 3.2, is perhaps the most well-known. Used in Afghanistan for a variety of missions, the Predator is a large, long-endurance UAV. It has a wingspan of 49 feet, a range of 400 nautical miles, and can carry

Figure 3.3: The Dakota unmanned aerial vehicle.

a payload of 450 lbs [76]. The Predator is commonly used for armed reconnaissance and target acquisition. An even larger UAV used by the military is the Global Hawk, with a wingspan of 116 feet and a range of 12,000 nautical miles [76]. One mid-size class UAV is the Dakota, shown in Figure 3.3. This platform is much less expensive than the long-endurance platforms of larger UAVs. The Dakota has a wingspan of 16 feet, a payload capacity of 80 lbs, and can fly for up to 6 hours [25]. Due to shorter ranges and smaller payload capacities, mid-size UAVs have more limited mission capabilities than larger UAVs. However, the cost per aircraft is much less. Micro UAVs (sometimes called mini UAVs or micro air vehicles) are even more cost effective than mid-size UAVs, but with a great trade-off in payload and range. One example of a micro UAV is the Dragon Eye, a UAV developed at the Naval Research Laboratory for use by Marines in reconnaissance. The Dragon Eye has a wingspan of 45 inches, a maximum payload of 1.8 lbs, and a maximum flight time just over one hour [14]. Micro UAVs might also have flight stability problems due to external disturbances such as wind gusts, since the size and weight of the UAVs are so small.

The UAV is becoming increasingly popular for many applications, particularly where high risk or accessibility are issues. Manned flights, regardless of the mission, place pilots in danger, but by using UAVs, the danger to humans is reduced. Manned aircraft are also generally much more expensive than unmanned craft, and because of this, UAVs can be used in nontraditional ways. For a large, expensive, manned aircraft, survivability is an essential mission goal. However, for a small, expendable, unmanned aircraft this is not always the case. If survivability is not a mission goal, the effective range of a UAV is increased, since it needn't plan to return to a recovery area. In general, larger UAVs like the Predator and Global Hawk are too expensive to lose during a mission; it is the mid-size and micro UAVs that offer the advantage of low cost and expendability. However, tradeoffs come with the lower cost. The smaller the UAV, the shorter the range and smaller the payload, decreasing the flexibility of the aircraft. To overcome this, smaller UAVs can be flown in groups with diverse and complementary abilities.

3.2.2 Applications

Both UAV technology and complementary subsystem technologies have progressed to the point where mid-size and micro UAVs can accomplish realistic and meaningful missions. UAVs are useful for a wide variety of applications, many of which are military in nature. UAVs are now considered sufficiently mature to serve as sensor and weapons platforms [48]. Smaller UAV platforms are particularly suited to radar jamming, surveillance, signals collection, data relay, and search.

A major advantage of the small size of a UAV is the ability to fly close to targets. Radar jamming benefits greatly from proximity to the target [14]. A common method for radar jamming is stand-off jamming, where the aircraft is outside the range of enemy weapons [2]. The power necessary to jam a radar at a given range is proportional to the square of the range between the radar and the jammer, so stand-off jamming requires large amounts of power [71]. For stand-in jamming, the jamming platform is within range of enemy weapons. The great advantage of stand-in jamming is the large decrease in the power necessary to effectively jam a radar. Because stand-in jamming places the jamming platform at risk, this technique is commonly used only for self-preservation in manned aircraft, since large planes can be easily tracked before jamming has begun. Small UAVs, however, have a great advantage in this type of mission. The ability of a radar to track an aircraft depends heavily on the radar cross section (RCS), the area of the aircraft visible to radar. Due to the small size of UAVs, the RCS tends to be very small, sometimes even similar to the size of a bird. This makes smaller UAVs more difficult to track than manned aircraft, opening the possibility for UAVs carrying jammers to fly very close to enemy radar. The combination of low radar signatures and expendability make UAVs excellent candidates for radar jamming applications.

Some other potential applications of UAVs are signals collection and data relay. Collecting information on electromagnetic signals is an important task for many military operations [14]. UAVs could record wireless audio communications from enemies on the ground, monitor communications from friendly vehicles in the area, or use communication data to track movement. In addition to collecting signals, UAVs could also function as data relays. A UAV able to communicate with a satellite or equipped with a long range transmitter could relay data between unmanned ground vehicles, other UAVs, and military personnel.

Search and surveillance are also excellent applications for UAVs. Surveillance from a UAV could take many forms. A UAV remotely controlled by a human could be equipped with one or more cameras and then flown for reconnaissance. The aircraft could either be fully controlled by a human or semi-autonomous, requiring the human to enter waypoints [14]. With the proper sensors, UAVs could also monitor an area for chemical or biological agents. In addition to human control, autonomous controllers can be designed for search and surveillance [79]. Search and surveillance place great importance on sensors for accomplishing a goal. For smaller UAVs, sensors must be light enough to meet payload limitations.

3.2.3 Operation

An important consideration for using UAVs is operation, as methods for generating plans to accomplish a desired task vary. Navigation of a UAV requires balancing a number of priorities, including safety, efficiency, and performance [23]. Control can be done entirely by a human, semi-autonomously with a human to set goals, or completely autonomously.

Direct human control is currently a common way to operate UAVs. Full control over the aircraft is retained, but there is no longer any danger to the human pilot. While human control of UAVs is generally as robust as human control over manned aircraft, there are a number of disadvantages associated with this control method. First, each aircraft requires a flight crew of at least one and up to three humans, which makes coordination of large swarms of UAVs just as difficult as coordination between conventional manned planes [68]. Communication channels to the aircraft must always be open in order for the remote pilot to maintain control over the UAV. For long range missions, this almost necessitates satellite communication, which takes up payload weight and consumes power.

Introducing some autonomous behavior can partially overcome these limitations. Semi-autonomous flight control allows the human to provide broad scale navigation planning, while a low level autopilot flies the UAV. Humans typically set waypoints along the UAV's path, and the autopilot flies between these points. While the immediate actions of the UAV might be autonomous, a human must still guide the aircraft. By making aircraft semi-autonomous, it is possible for one human to control multiple UAVs.

For situations where no human has complete information, fully autonomous control may be preferable to human guided control. While a human may set goals for the UAV, both high and low level flight control are done autonomously. With methods for flight coordination, large numbers of autonomous controllers can be deployed in a single mission. Coordination between

UAVs can be included in the autonomous controllers. While coordination is more feasible with humans out of the loop, hand programming controllers for multi-UAV systems remains difficult. Autonomous controllers can also be very reactive to the environment, something that may be difficult for remote human operation. Automatic methods of developing autonomous controllers – like evolutionary computation – have great potential to make controller development easier.

3.2.4 Controller Architecture

The controller architecture of the flight system for this research is divided into two parts: navigation and low level control. Only the navigation portion of the flight controller is evolved; the low level flight control is done by an autopilot. The navigation controller receives radar electromagnetic emissions as input, and based on this sensory data and past information, the navigation controller changes the desired roll angle of the UAV control surface. The autopilot then uses this desired roll angle to change the heading of the UAV. This autonomous navigation technique results in a general controller model that can be applied to a wide variety of UAV platforms; the evolved controllers are not designed for any specific UAV airframe or autopilot.

3.2.5 Simulation Flight Model

A model approximating the flight of a UAV is used in the simulation [80]. At every time step, the position and orientation of the UAV is updated based on the roll angle, the single output variable available to the navigation controller. For this research, each time step is 1 second long. The speed of the UAV is fixed at 80 knots and the altitude is held constant. At each time step, after the new roll angle has been set by the navigation controller, the simulation updates the position of the UAV. First the roll angle is used to calculate a turn rate. The turn rate is used to calculate the new UAV heading, which is then used to find the new position of the UAV.

3.3 Radar

The goal of this research was to evolve UAV controllers to fly to radars and circle closely around the radar sites. The primary application of this type of controller is radar jamming, though a UAV equipped with this type of controller could be used for reconnaissance or even to carry weapons to attack a hostile radar.

3.3.1 Radar Types

The simulation can model a wide variety of radars. The site type, emitter function, frequency, gain, noise, power, pulse compression gain, bandwidth, minimum emitting period, mean emitting period, minimum emitting duration, and mean emitting duration are all configurable in the simulation. For the purposes of this research, most of these parameters were held constant. Radars used in experiments can be described based on a few characteristics of the radar: 1) site type, 2) emitter function, and 3) emitting pattern. The site type of the radar determines the target radar's mobility.

The available site types are stationary sites and mobile sites; the latter models a surface-to-air missile (SAM) site. The stationary site has a fixed position for the entire simulation period. The mobile site is modeled by a finite state machine with the following states: *move*, *setup*, *deployed*, and *tear down*. When the radar moves, the new location is random, and can be anywhere in the simulation area. The finite state machine is executed for the duration of simulation. The radar site only emits when it is in the *deployed* state; while the radar is in the *move* state, the UAV receives no sensory information. The time in each state is probabilistic, and the minimum amount of time spent in the deployed state is an hour, 25 percent of the total simulation time.

Two emitter functions are used in this research. The first type is early warning (EW) radar, a generally low-resolution, low-frequency search radar. This is a common emitter type for stationary radars. The second type is target acquisition (TA), an emitter function used by surface-to-air missiles to find targets [71]. SAM sites have both TA and target tracking (TT) radar, both of which are modeled in the simulation, but TT radar was not used to evolve controllers. Since TT radar is generally only used once TA radar has acquired a target, TA radar is a more reliable way to track a SAM site. In this research, all stationary radars use EW radar and all mobile radars use TA radar.

The last characteristic used to describe radars in this research is the emitting pattern of a target radar site. Radars can emit continuously, intermittently with a regular period, or intermittently with an irregular period. This aspect of the radar is configured by setting the minimum emitting period, mean emitting period, minimum emitting duration, and mean emitting duration. If all four parameters are set to infinity, the radar is continuous. If the minimum and mean are the same for both period and duration, then the radar is considered to be emitting with a regular period. If the minimum and mean are different, the radar is emitting with an irregular period.

3.3.2 Radar Modeling

Almost all radar antennas are directive, focusing most of the transmitting power into the main beam. The main beam, which tends to be narrow for directive radar, is the portion of the emitted signal used by the radar for detection and tracking. The sidelobes are the parts of the emitted signal that are not part of the main beam [74]. The main beam is the effective portion of the radiation emitted by the radar, so sidelobes have a much lower power than the main lobe, making them harder to detect. However, sidelobes exist in all directions, while the main beam is visible only in the direction in which the radar is pointed. For this reason, if a controller can track a radar based only on the sidelobes, the radar can be tracked no matter the direction in which it is pointed. The simulation for this research models only the sidelobes of the radar; this model is intended to increase the robustness of the system, so that the controller doesn't need to rely on a strong signal from the main beam.

While the simulation area includes no other sources of radiation other than the radar, real radars operate in an environment with many external sources of noise. Radars themselves produce noise, such that the emitted signal is not always perfectly ideal. When simulating a radar, noise should be included in the model. In this simulation, Gaussian noise is added to the amplitude of the radar signal. This helps to model the noise a sensor would pick up when sensing a radar signal. In addition to making the simulation more realistic, the addition of noise to the simulation environment helps to promote the evolution of robust controllers [33].

3.4 Sensors

A key component of the UAV system is the receiving sensor. In this research, the sensor detects electromagnetic signals from radars in the simulation. The receiving sensor can perceive only two pieces of information: the amplitude and the angle of arrival (AoA) of incoming radar signals. These two sensor values and the slope of the amplitude are available to the controller. The slope of the amplitude is found by computing a quadratic least squares fit of the amplitude over the previous five minutes.

The AoA measures the angle between the heading of the UAV and the source of incoming electromagnetic energy (as shown in Figure 3.4). Real AoA sensors do not have perfect accuracy

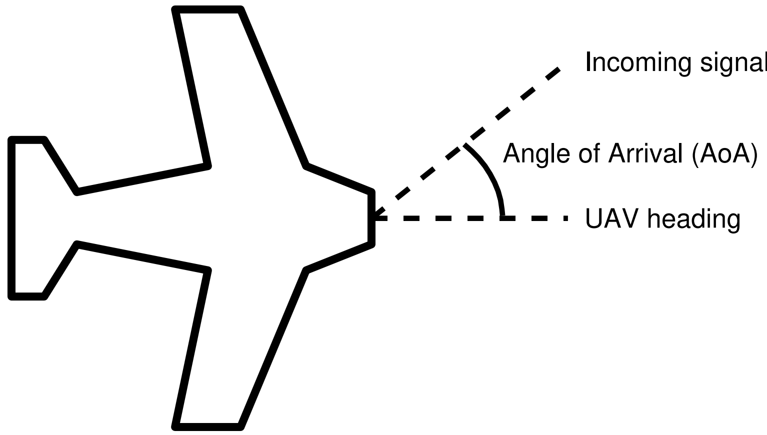


Figure 3.4: The angle of arrival (AoA) is the angle between the UAV heading and the incoming signal.

in detecting radar signals, so the simulation models an inaccurate sensor. The accuracy of the AoA sensor can be set in the simulation. In the experiments described in this research, the AoA is accurate to within $\pm 10^\circ$ at each time step, a realistic value for this type of sensor.

3.5 Problem Difficulty

While a human could design a controller capable of homing in on a radar under perfectly ideal conditions, the real-world application for these controllers is far from ideal. As noise increases, this problem becomes much more difficult for humans to solve well. While sensors to detect the amplitude and angle of arriving electromagnetic signals can be very accurate, the more accurate the sensor, the larger and more expensive it tends to be. One of the great advantages of UAVs is their low cost, and the feasibility of using UAVs for many applications may also depend on keeping the cost of sensors low. By using evolution to design controllers, cheaper sensors with much lower accuracy can be used without a significant drop in performance. As the accuracy of the sensors decreases and the complexity of the radar signals increases – as the radars move or emit periodically – the problem becomes far more difficult for human designers.

This research is interested in evolving controllers for these difficult, real-world problems.

3.6 Transference to real UAVs

Transference of controllers to a real UAV is an important issue. Evolved controllers could exhibit excellent behavior in simulation, but if these same controllers fail to perform well in the real world, then evolution can hardly be considered a success. Flying a physical UAV with an evolved controller is planned as a demonstration of the research, so transference was taken into consideration from the beginning. Several aspects of the controller evolution were designed specifically to aid in this process. First, navigation was abstracted from direct control of the UAV's flight. Second, simulation was tuned for equivalence to real UAVs. Third, noise was added to the simulation to encourage the evolution of robust controllers.

Rather than attempting to evolve direct control, only navigation was evolved. The navigation control was abstracted from the flight of the UAV both to aid in transference of controllers to real UAVs and to make evolved controllers applicable to a variety of airframes. By modeling only navigation, the simulation did not need to include a specific, accurate model. If direct control had been evolved, an in-depth model of a specific airframe would have had to be included, making development of the simulation environment more difficult and limiting the applicability of evolved controllers. By using an autopilot for the low level control and only evolving the high level navigation control, evolved controllers could be used for different UAV airframes. Using an autopilot capable of low level control of a particular airframe, the evolved controller can manage the navigation of the UAV.

The simulation parameters were designed to be tuned for equivalence to real aircraft. While the use of navigation control allowed a general model of the UAV in simulation, the adjustment of some parameters could aid in transference. For example, the simulated UAV is allowed to update the desired roll angle once per second reflecting the update rate of the autopilot controlling a real UAV being considered for flight demonstrations of the evolved controller. For autopilots with slower response times, this parameter could be increased. The speed of the UAV in simulation could also be adjusted to correspond to the speed of a particular UAV.

Noise was added to the simulation to encourage the evolution of robust controllers. The addition of appropriate levels of noise to a simulation was shown in [33] to aid in the transference of evolved controllers to real robots. By adding noise to the emitted radar signal and modeling sensor inaccuracies, the problem solved by evolution becomes more realistic, and evolved controllers are better equipped to respond to noise in the real world. A noisy simulation environment encourages the evolution of robust controllers that are more applicable to real UAVs.

Chapter 4

Evolution and Fitness Evaluation

The goal of this research was to use genetic programming (GP) to evolve navigation controllers for unmanned aerial vehicles. During each evolutionary run, GP evolved a population of programs to fly a UAV to a radar and circle closely around it. During flight, the UAV was also supposed to maintain an efficient flight path by flying with the wings level to the ground for as much time as possible and by avoiding sudden, sharp turns.

To use GP, a human designer must make choices about several aspects of the evolutionary algorithm that will be used. The parameters of GP are configurable, and the human designer selects parameters that will encourage evolution to find an appropriate solution. The types and frequencies of crossover and mutation, the method of selection, the method of evaluation, and many other parameters of GP are all chosen before evolution begins. Many of these parameters can be selected based on previous research.

Another area of the GP algorithm where the human designer must make choices is in the selection of functions and terminals. Every individual program in a GP population is represented as a tree, where non-leaf nodes are functions and leaf nodes are terminals [36]. Terminals are commands that take no arguments, while functions take at least one argument. The number of arguments that a function takes equals the number of children for corresponding node. The human designer provides these basic lists of commands that an evolved program is built from.

The human designer also chooses an appropriate method for measuring the fitness of an individual. The formulation of fitness functions has always been one of the major difficulties of EC research [5]. For many problems explored to date in ER, fitness functions that combined multiple objectives were synthesized using extensive human knowledge of the domain or trial and error. For proof of concept research, the problem to be solved has often been adapted in ways that made the formulation of these fitness metrics easier, such as the simplification of the environment [57]. Co-evolution and competitive fitness metrics have been used to generalize fitness function formulation, but these methods usually require changing the problem to fit the competitive fitness model [58, 67]. For problems without a single, easily quantifiable objective, an alternative that has attracted a great deal of research in the last several years is multi-objective optimization, which allows the evolutionary algorithm to optimize on multiple fitness metrics [13]. Multi-objective genetic programming with four fitness functions is used in this research.

A variety of strategies for evolving robust programs have been investigated. These strategies are intended to address difficulties like the bootstrap problem where no appreciable fitness exists in early populations, premature convergence on a suboptimal solution, or problems too large for direct evolution on a final fitness function. Techniques to address these problems include competitive co-evolution, incremental evolution, multiple populations, steady-state evolution, and many other methods. Incremental evolution addresses many difficulties of evolution by starting with a simple problem and increasing the difficulty incrementally. In this work, two types of incremental evolution, functional and environmental, are used to aid evolution.

In this chapter, the evolutionary system used in this research is presented. In Section 4.1, the multi-objective genetic programming algorithm used to evolve controllers is described. The parameters of the algorithm, lists of functions and terminals, and the method of evaluation are all given. In Section 4.2, the four fitness functions are described. The method for combining the four fitness functions is also described. In Section 4.3, incremental evolution techniques used in the research are presented. Explanations of both functional and environmental incremental evolution methods are described.

4.1 Multi-objective Genetic Programming

Multi-objective optimization has been growing in popularity and usefulness recently [13], but it has only begun to be applied to GP in the last several years [69]. Multi-objective optimization allows evolution to use multiple fitness functions. Unlike traditional EC, where individuals in a population can be ranked in order, multi-objective optimization uses a non-dominated sort to rank individuals. In addition to the relative rank of the individual, multi-objective algorithms often measure the diversity of individuals in the population using crowding distance. The multi-objective genetic programming algorithm used by the evolutionary process in this research is very similar to NSGA-II, developed by Deb, Agrawal, Pratap, and Meyarivan [15].

4.1.1 Genetic Programming Parameters

When using GP, the parameters of the algorithm must be chosen so that evolution has a good chance of producing fit controllers. The speed of the algorithm and methods of evaluation must also be taken into account. Parameters used in this research, shown in Table 4.1, were chosen with these issues in mind.

One parameter of GP is whether evolution is generational or steady-state. While generational evolution is more common, steady-state evolution is widely used. In this research, GP evolved

Table 4.1: Genetic programming parameters

Parameter	Value
Generations	600
Population Size	500
Initialization Scheme	Ramped half-and-half
Selection Scheme	Tournament
Maximum Initial Depth	5
Maximum Depth	21
Crossover Rate	0.9
Mutation Rate	0.05
Tournament Size	2
Trials per evaluation	30

each population of 500 individuals for 600 generations. One reason for using generational evolution with a fixed number of generations is because this problem has no clear cutoff where evolution can be stopped.

When creating a new GP population, a variety of initialization methods may be employed. The two basic methods for generating random program trees are the "full" method and the "grow' method. In the "full" method, the path between the root node and every leaf node is the same. A "full" tree is generated by selecting random members of the function set to fill all nodes until the desired depth is reached, when nodes are selected from the terminal set. Individuals in populations initialized using the "full" method can all have the same depth or the depth can be ramped. When ramping is used, there are an equal number of trees of each depth between 2 and the maximum. In the "grow" method, the maximum depth of the tree is the same, but the paths from the root node to all leaf nodes are not equal. The "grow" method creates variably shaped program trees. A "grow" tree is generated by selecting a random member of the union of the function and terminal sets for each node until the maximum initial depth is reached. This initialization function can also be ramped. The preferred method for GP is the "ramped half-and-half" method. This method combines the "full" and "grow" methods, creating a wide variety of tree sizes and shapes. Half of the population is initialized using the “ramped full” method and the other half is initialized using the “ramped grow” method. The “ramped half-and-half” method is used in this research.

To use these generative methods, a maximum initial depth must be specified. Ramping occurs between the minimum depth of 2 and the maximum, which in this research is set to 5. Code growth, or bloat, is a common phenomenon in variable-length genome EC research like GP [6, 75]. To prevent trees from growing out of control, a maximum depth limit is set. In this research, the maximum tree depth was 21.

Selection is an extremely important step in the evolutionary process, and there are a variety of selection methods. In fitness-proportionate selection, the probability of selecting a given individual is based on its fitness and the total fitness of the population. This method generally requires sorting which is computationally expensive. Also, when fitness values are close together, this method is not much better than other methods. Another selection method is rank selection. In this method, members of the population are sorted into ranks of fixed size, with the probability of selection based on the rank the individual is in. This method works well when fitness values are clumped together, helping to prevent premature convergence, but it still requires sorting. The method used in this research is tournament selection. In this method, several individuals are selected at random from the population and the individual with the best fitness value is selected. Tournament selection functions as probabilistic rank selection, reducing the importance of sorting the population. A tournament size of 2 is used in this research.

The primary genetic operator used in GP is crossover, also called recombination. This operation starts with two parent individuals and produces two children. Once two parents have been selected from the population, one node in each parent is randomly selected as the crossover point. The subtrees with each of these nodes for roots are then swapped. Figure 4.1 demonstrates this process. Figure 4.1a and Figure 4.1b show two parent program trees with the

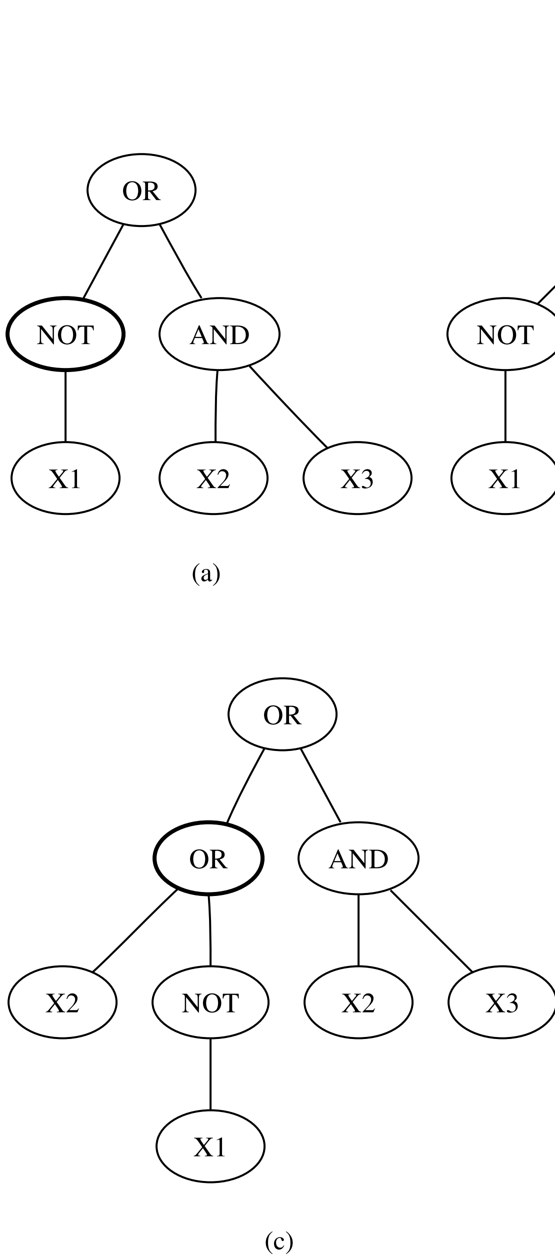


Figure 4.1: An example of the recombination process, with the crossover points highlighted. Two parent program trees (a and b) produce two children (c and d).

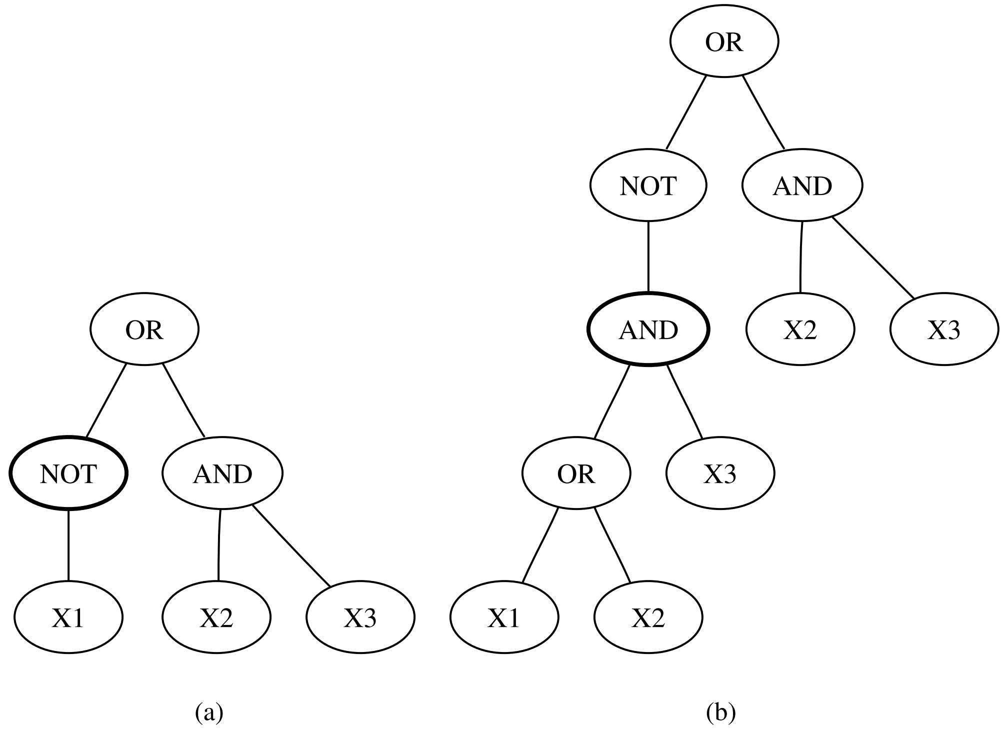


Figure 4.2: An example of the mutation process, with the mutation point highlighted. A parent tree (a) produces a mutated child tree (b).

crossover points highlighted. Figure 4.1c and Figure 4.1d show the children that result from crossover. As long as the resulting subtrees are not deeper than the maximum depth allowed, the children are added to the new generation of the population. The crossover method used in this research is equivalent to the method described by Koza [36]. The crossover rate, the probability of performing crossover as a genetic operator, is 0.9 in this work.

The other genetic operator used by GP is mutation. Mutation is used much less frequently than crossover and is primarily intended to introduce diversity into the population. The mutation operator starts with a single parent and produces a single child. After the parent is selected from the population, a node is randomly selected as the mutation point. The subtree rooted at this point is destroyed, and a randomly generated subtree is inserted at that node. Figure 4.2 demonstrates this process. Figure 4.2a shows a parent program tree with the mutation point highlighted. The subtree rooted at that node is destroyed, and a random subtree is generated and placed at that point. The child produced by mutation, shown in Figure 4.2b, is then inserted into the new population. Like crossover, the mutation method used in this work is equivalent to the method described by Koza [36]. The mutation rate is 0.05 in this research.

The individuals in the population all represent UAV controllers, so to evaluate the population it is necessary to simulate the flight of each controller. Since the initial conditions of the simulation are random, it is desirable to evaluate each controller over a number of simulated trials. While the results are more accurate the greater the number of trials, performing a large number of simulations is computationally expensive, so there must be some balance between the two. In this research, each individual is evaluated over 30 trials.

4.1.2 Functions and Terminals

The function and terminal sets used in this research combine a set of very common functions used in GP experiments and some functions specific to this problem. The function set is defined as:

$$F = \{ \textit{Prog2, Prog3, IfThen, IfThenElse, And, Or, Not}, <, \leq, >, \geq, < 0, > 0, =, +, -, *, \div, X < 0, Y < 0, X > max, Y > max, \textit{Amplitude} > 0, \textit{AmplitudeSlope} > 0, \textit{AmplitudeSlope} < 0, \textit{AoA} > 0, \textit{AoA} < 0 \}$$

The *Prog2* and *Prog3* functions take two and three arguments respectively. Both of these functions simply execute all of the arguments in order. The *IfThen* and *IfThenElse* functions act as conditionals. The *IfThen* function takes two arguments; if the first argument is true, the other argument is executed. The *IfThenElse* function takes three arguments; if the first argument is true, the second arguments is executed, otherwise, the third argument is executed.

The *And*, *Or*, and *Not* functions are logic functions with two arguments each. The relation arguments $<, \leq$, $>$, $\geq$, <0 , >0, and = as well as the arithmetic arguments +, -, *, and ÷ all take two arguments.

The UAV has a global positioning system (GPS) on-board, and the position of the UAV is given by the x and y distances from the origin, located in the southwest corner of the simulation area. This position information is available using the functions that include *X* and *Y*, with *max* equal to 100 nmi, the length of one side of the simulation area. The radar is always placed inside the simulation area, but the UAV is free to move beyond it. The $X < 0$, $Y < 0$, $X > max$, and $Y > max$ functions act as conditionals. For each function, the single argument is executed if the condition is true.

The two available sensor measurements are the amplitude of the incoming radar signal and the AoA, or angle between the heading and the source of incoming electromagnetic energy. Additionally, the slope of the amplitude with respect to time is available to GP. The *Amplitude > 0, AmplitudeSlope > 0, AmplitudeSlope < 0, AoA > 0,* and *AoA < 0* functions act as conditionals with a single argument.

The terminal set is composed of commands that take no arguments, and is defined as:

> *T = { HardLeft, HardRight, ShallowLeft, ShallowRight, WingsLevel, NoChange, rand, 0, 1 }*

When turning, there are six available actions. Turns may be hard or shallow, with hard turns (*HardLeft, HardRight*) making a $10°$ change in the roll angle and shallow turns (*ShallowLeft, ShallowRight*) a $2°$ change. The *WingsLevel* terminal sets the roll angle to 0, and the *NoChange* terminal keeps the roll angle the same as before this command was executed. Multiple turning actions may be executed during one time step, since the roll angle is changed as a side effect of each terminal. The final roll angle is passed to the autopilot after execution of the navigation controller. The maximum roll angle is $45°$. Each of these six terminals returns the current roll angle. The final three terminals, *rand*, *0*, and *1* have no side effects, the commands simple return a value. The value of the *rand* terminal is fixed for a given node when that node is created.

4.1.3 Parallel Evaluation

In GP, evaluating the fitness of the individuals within a population takes significant computational time. The evaluation of each individual requires multiple trials, 30 trials per evaluation in this research. During each trial, the UAV and the radar are placed randomly and four hours of flight time are simulated. Evaluating an entire population of 500 individuals for a single generation requires 15,000 trials. Therefore, using massively parallel computational processors to parallelize these evaluations is advantageous.

Many methods of parallel computing exist, but two methods are widely used in EC research done with parallel computers. The *island* model of parallel processing has been used with success by Koza [38] and others. Traditionally, GP uses a single population, but for parallel computation, splitting the total population in sub-populations, one per computer, helps to reduce the amount of communication necessary. Every computer in a cluster has an independent GP system, and individuals can migrate between computers.

Another popular method of parallel computation employs the concept of master and slave nodes. One computer in a parallel cluster is designated as the master node, and this computer manages the evolutionary process. The evaluations are the computationally expensive portion of GP, so the master node distributes individual evaluations among the other computers in the cluster, which are designated as slave nodes. Once a slave node has completed an evaluation, the results are returned to the master node. After the master node collects all the fitness values associated with each individual evaluation from the slave nodes, GP moves to the selection process. The *master-slave* parallel model allows the traditional single population model to be used while benefiting from the increased speed of parallel processing.

In this research, the master-slave model of parallel processing was used. The data communication between master and slave processors was done using the Message Passing Interface (MPI) standard [66] under the Linux operating system. The master node ran the GP algorithm and did all computations related to selection, crossover, and mutation. Evaluations of individuals in the population were sent to slave nodes for processing. The parallel computer used for the experiments was a Beowulf cluster made up of 46 computers running Red Hat Linux. Each computer had two 2.4 GHz Pentium 4 processors with hyper-threading, for a total of 92 processors in the cluster. Hyper-threading provides a small performance gain for multiple simultaneous processes, so two slave nodes were run on each processor, for a total of 184 slave nodes spread over the 92 processors in the cluster.

4.2 Fitness Functions

Four fitness functions determine the success of individual UAV navigation controllers in this work. The fitness of a controller was measured over 30 simulation trials, where the UAV and radar positions were different for every run. The four fitness measures were designed to satisfy the four goals of the evolved controller: moving toward the emitter, circling the emitter closely, stable flight, and efficient flight.

4.2.1 Normalized distance

The primary goal of the UAV in this research is to fly from its initial position to the target radar site as quickly as possible. To develop a fitness function that could effectively promote this behavior, the average squared distance between the UAV and the radar is measured. The intent of this fitness function is to reward controllers that fly toward the radar as directly as possible and to punish those that take less direct paths or fly away from the radar.

The random placement of both the UAV and the radar for each simulation poses a slight problem in developing fitness functions to measure this behavior. The fitness values for cases where the initial distance between the radar and the UAV is large will be much greater than for cases where this initial distance is small. When averaging these fitness values, those cases with large initial distances will come to dominate the final fitness value. To counter this, the distance between the radar and the UAV is normalized using the initial distance. Some small biases do still exist. For large initial distances, it still takes longer for the UAV to reach the radar than for shorter initial distances, and for shorter initial distances, small deviations have greater weight in the final fitness value than for cases with large initial distances.

The normalized distance fitness measure is given as the average over all time of the square of the distance between the UAV and the radar for each time step divided by the initial distance between the UAV and the radar

$$fitness_1 = \frac{1}{T}\sum_{i=1}^{T}\left[\frac{distance_i}{distance_0}\right]^2 \tag{4.1}$$

where T is the total number of time steps, $distance_0$ is the initial distance, and $distance_i$ is the distance at time i. Evolution is trying to minimize $fitness_1$.

While this fitness function does have some minor biases, evaluating each controller over a large number of trials helps to mitigate these biases. Since placement of radars and UAVs for each trial is random, the evaluation for a single controller includes a variety of initial distances. Rather than focusing on optimization, this fitness measure was designed to promote the evolution of a particular behavior in the controllers being evolved by GP.

4.2.2 Circling distance

The second major goal of the UAV controller in this research is to circle around the radar once the UAV has arrived in the radar's vicinity. Many of the potential applications of this type of controller benefit heavily from a small circling distance. In particular, radar jamming requires much less power when the jammer is close to the source, since the power required for effective jamming increases proportional to the square of the distance. To promote the evolution of tight circling behaviors in controllers, a circling distance fitness function was designed.

The normalized distance fitness function ($fitness_1$) does measure circling behavior, but because circling is such a small percentage of the initial distance between the UAV and the radar, circling has only a minor influence on the final fitness value, and very little evolutionary pressure is applied to develop good circling behavior. The addition of a separate circling fitness metric helps to place more emphasis on this aspect of the controller's behavior. In this function, an arbitrary distance much larger than the desired circling radius is defined as the in-range distance. For this research, the in-range distance was set to 10 nmi. The circling distance fitness metric measures the average distance between the UAV and the radar over the time the UAV is in-range.

The circling distance fitness function is given as the average over the time the UAV spends in-range of the radar of the square of the distance to the radar

$$fitness_2 = \frac{1}{N} \sum_{i=1}^{T} in\_range * (distance_i)^2 \tag{4.2}$$

where *T* is the total number of time steps, *N* is the amount of time the UAV spent within the in-range boundary of the radar, $distance_i$ is the distance at time *i,* and *in_range* is 1 when the UAV is in-range and 0 otherwise. Evolution is trying to minimize $fitness_2$.

It was not necessary to normalize the circling distance fitness function because this function only applies when the UAV is in-range of the radar. There are no biases associated with the initial distance between the UAV and the radar. This fitness metric effectively promotes the evolution of controllers that circle tightly around a radar site.

4.2.3 Level time

Flying to the radar and then circling closely around it are the primary goals of the UAV controller. For a controller designed to work only in simulation, the first two fitness functions would probably be sufficient. However, the controllers evolved in this research were intended for use on real UAVs, which have less ideal flight characteristics than simulated aircraft. In simulation, UAVs can be flown with any control scheme that accomplishes the primary tasks, but physical UAVs do not respond well to frequent and drastic changes in roll angle. This manner of control can create dangerous flight dynamics, reduce the flight range of the UAV, or even lead to crashes.

The first of two fitness metrics that measure the efficiency of the flight path is the level time, the amount of time the UAV spends with a roll angle of $0°$. This is the most stable flight position for a UAV, and the UAV should spend as much time as is feasible at this roll angle. The level time fitness measure applies in opposite situations from the circling distance fitness function. The circling distance metric applies only when the UAV is in-range of the radar, but the level time fitness metric only applies when the UAV is outside the in-range distance. While the UAV is not in-range, it should fly straight, but once the UAV is in-range, it should circle around the radar, which requires the UAV to increase the magnitude of its roll angle.

The level time fitness function is given as the number of time steps that the UAV spends level out of range divided by the total time spend out of range

$$fitness_3 = \frac{1}{T - N} \sum_{i=1}^{T} (1 - in\_range) * level \tag{4.3}$$

where *T* is the total number of time steps, *N* is the amount of time the UAV spent within the in-range boundary of the radar, *in_range* is 1 when the UAV is in-range and 0 otherwise, and *level* is 1 when the roll angle has been equal to $0°$ for two consecutive time steps and 0 otherwise. Evolution is trying to maximize $fitness_3$.

This fitness measure promotes the evolution of controllers that change roll angle infrequently. The level time fitness function is designed to work in cooperation with other fitness measures. If a controller was optimized only for this function, the ideal behavior would be to fly straight and level for the entire four hour period, not at all the desired behavior for a UAV controller. The conflict between maximizing this function and satisfying the other functions is why multi-objective optimization is so well suited to this problem.

4.2.4 Turn cost

A fitness function that encourages keeping a roll angle of $0°$ helps to promote the evolution of controllers that exhibit efficient flight paths, but it does little to discourage the common and potentially dangerous control scheme of frequent roll angle changes. A very simple way to home in on a radar is to try to always keep the AoA at $0°$. However, because the AoA sensor is noisy and inaccurate, this control scheme would require changing the roll angle at every time step, often by large amounts. While UAVs are capable of very quick, sharp turns, it is preferable to avoid them in real flight. Switching between right and left banking turns every second could lead to an unstable flight and a possible crash.

To avoid rapid, drastic changes in the roll angle, a measure of the turn cost is used as the last fitness function. Turns that contribute to this fitness function must be large; small turns have less potential harm. The turn cost fitness measure is intended to penalize controllers that navigate using a large number of sharp, sudden turns. The UAV can achieve a large turning radius without penalty by changing the roll angle gradually; this fitness metric only accounts for cases where the roll angle has changed by more than 10° since the last time step.

The turn cost is given as the average over all time of changes in roll angle over 10°

$$fitness_4 = \frac{1}{T}\sum_{i=1}^{T} h\_turn * |roll\_angle_i - roll\_angle_{i-1}| \tag{4.4}$$

where *T* is the total number of time steps, *roll_angle* is the roll angle of the UAV and *h_turn* is 1 if the roll angle has changed by more than 10° since the last time step and 0 otherwise. Evolution is trying to minimize $fitness_4$.

By penalizing drastic turns, the turn cost fitness function promotes controllers that use shallow turns rather than large, sharp turns. This fitness measure does not forbid controllers from using large roll angles, it simply encourages gradual changes in the roll angle over time. Optimizing this fitness measure is only worthwhile if the other functions are also successful.

4.2.5 Combining the Fitness Measures

Each of the four fitness functions was designed to evolve a particular behavior. The normalized distance fitness function was intended to produce controllers that would move quickly to a radar. The circling distance fitness function was designed to promote tight circling behavior. The level time fitness function was intended to encourage changing the roll angle infrequently and flying level for long periods. The turn cost fitness function was designed to discourage large changes in roll angle. Each of these desired behaviors could be identified and quantified by human designers.

What is most encouraging about these fitness functions is that it isn't particularly of interest to evolve controllers that optimize one of the fitness functions. A good controller should have evolved all of these behaviors, and success at one shouldn't come at cost to another.

Multi-objective optimization seemed a natural method for combining these fitness functions. Using some weighted function that produced a single fitness value from the four fitness functions would have been preferable if some ratio of importance between the functions was known *a priori*, but this was not the case. Finding a Pareto front of solutions using a non-dominated sort as part of multi-objective optimization was far more attractive.

Applying the term multi-objective optimization to this evolutionary process is a misnomer, because this research was concerned with the generation of behaviors, not optimization. In the same way that a traditional genetic algorithm can be used for both optimization and generation, so can multi-objective optimization. When considering this evolutionary system, the distinction between optimization and generation should be taken into account. Also, even though the controller doesn't generate the most optimized controllers possible, it can obtain near-optimal solutions.

4.3 Incremental Evolution

Many techniques exist – with varying levels of success – for overcoming problems associated with evolving solutions to difficult problems. As discussed earlier, the bootstrap problem, where no individuals in early populations have measurable levels of fitness, can be a major problem in EC. Another major problem is the inability of evolution to overcome local fitness peaks, where a population converges quickly on a sub-optimal result and is unable to make further progress. Many of the techniques used in EC to counter these problems are discussed in Section 2.3.

In this research, incremental evolution is used to aid evolution in designing robust and fit controllers. In traditional direct evolution, a randomly initialized population is directly evolved on the final task problem. In incremental evolution, the initial population is evolved on a simpler problem, and the resulting population is used as a seed for evolution on a more difficult problem. Evolution can proceed directly to the final task or use a series of increments.

There are two types of incremental evolution. Functional incremental evolution changes the difficulty of the fitness function in order to increase the difficulty of the problem. Environmental incremental evolution changes the environment to increase difficulty without changing the fitness function. These two types of incremental evolution are described in Section 2.3.5. Both types of incremental evolution were used in this research.

Maintaining sufficient diversity in the population is often an issue when using incremental evolution [18]. If the diversity of a population decreases too much during an early stage of evolution, the final evolution might still have a very difficult time producing a good solution. While this was always a concern in this research, one of the features of the multi-objective optimization algorithm had potential to counter loss of diversity. Like NSGA-II [15], the algorithm used for this research attempts to spread solutions across the Pareto front by incorporating a crowding distance into fitness evaluation, encouraging diversity in the population.

4.3.1 Functional Incremental Evolution

Functional incremental evolution incrementally changes the fitness function to increase the difficulty of the problem. A simple form of functional incremental evolution was used in this work. Controllers were evolved for 600 generations, and for the first 200 generations, only one of the four fitness functions was used. Of the behaviors the fitness functions were trying to evolve, flying to the goal was the most basic and most important. To place more importance on this behavior, only the normalized distance fitness function was used for the first 200 generations. For the last 400 generations, all four of the fitness functions were used.

While some doubt has been raised about using functional incremental evolution, for this problem it works well and makes a great deal of sense. The use of multi-objective optimization for the majority of evolution means that the fitness evaluation does not change completely, but only becomes more difficult as the additional three fitness functions are added. For this research, it is also very clear that this is the most important behavior to evolve and the simplest.

This use of incremental evolution is aimed at overcoming the bootstrap problem. The most important goal of the UAV controller is flying to the radar, but early generations can be very unsuccessful at this task. A controller might have very good scores for the level time and turn cost fitness functions and still be completely inept, so it is important to bring the population to a higher level of competency before introducing those fitness functions. The use of functional incremental evolution helps to overcome this problem.

4.3.2 Environmental Incremental Evolution

Environmental incremental evolution incrementally increases the difficulty of the environment or task faced by evolution, while leaving the fitness function unchanged. This second type of incremental evolution is also used in this research to boost the success of evolution in producing fit controllers for the more difficult radar types. First, controllers are evolved for continuously emitting, stationary radars. This baseline radar type is the easiest of the types used in this research for evolution to handle. The resulting population is then used as a seed for more difficult radar types.

In the simplest case, environmental incremental evolution is done in two stages. A new random population is initialized and then evolved for 600 generations on continuously emitting, stationary radars. Then, the resulting population is used as the initial population for a new evolutionary run. This second run is evolved for 400 generations using a different type of radar, like an intermittently emitting, stationary radar or a continuously emitting, mobile radar. The seed population is not immediately able to solve this new problem well, but since many aspects of the problem are similar, the seed population provides an excellent basis for evolving fit controllers for the new task.

Additional stages can be added to increase the effects of incremental evolution. For example, after the first stage of evolution on a continuously emitting, stationary radar, a population might be evolved on a continuously emitting, mobile radar and then on an intermittently emitting, stationary radar. For the most difficult radar types, such as the intermittently emitting, mobile radar, incremental evolution can be done in many stages, with multiple increments on both mobile and intermittent radars. This technique can also be used to develop a single controller that is capable of handling all radar types.

Chapter 5

Experiments and Results

In this chapter, experiments using the evolutionary system described in Chapter 4 are presented. These experiments were designed to test the ability of the multi-objective genetic programming system to evolve controllers for UAVs. Controllers were evolved on radar types of varying difficulties.

The four fitness functions used in this research were designed based on the behaviors desired in the final controller. In order to test the impact of each of the fitness functions, subsets of the four fitness functions were used to evolve controllers for the simplest radar type, a continuously emitting, stationary radar. The behaviors of controllers evolved using less than four fitness functions were compared with those of controllers evolved with all four functions. A comparison was also made between a human-designed controller and an evolved controller. These experiments are described in Section 5.1.

While traditional evolution produces a single best individual that can be compared with the best individual from a separate evolutionary run, multi-objective optimization produces a Pareto front of solutions, making comparisons between individuals more difficult. In order to gauge the performance of evolution, a method was devised to measure a controller's performance on the final task. This metric is described in Section 5.2.

Sections 5.3 and 5.4 present the evolutionary experiments performed in this research. The first set of experiments, presented in Section 5.3, directly evolved controllers for a variety of radar types from random initial populations. The second set of experiments, presented in Section 5.4, evolved controllers using incremental evolution. Experiments used five radar types: 1) continuously emitting, stationary radars, 2) intermittently emitting, stationary radars with regular periods, 2) intermittently emitting, stationary radars with irregular periods, 4) continuously emitting, mobile radars, and 5) intermittently emitting, mobile radars.

Though the UAV navigation controllers were evolved in simulation, the major goal of the research is using these controllers to fly real UAVs. The transference of controllers from simulation to the real world was an important factor in the simulation design. Flight tests of UAVs using these controllers are planned, but to test the real-world performance of the controllers in the near term, evolved navigation controllers were transferred to a wheeled mobile robot. In Section 5.5, the results of this experiment are presented.

5.1 Effectiveness of Fitness Functions

Based on initial results from evolution, it appeared that the four fitness functions selected for this research were effective in producing controllers with all the desired behaviors. However, since the individual impacts of each of the functions was unknown, it was difficult to tell how necessary all four functions were to the success of evolution. To test the effectiveness of each of the four fitness measures, evolutions were run with various subsets of the fitness metrics. These tests were done using the stationary, continuously emitting radar, the simplest of the three radar types presented above. The first fitness measure, the normalized distance ($fitness_1$), was included in every subset.

When only $fitness_1$ was used to measure controller fitness, flight paths were very direct. The UAV flew to the target in what appeared to be a straight line. To achieve this direct route to the target, the controller would use sharp and alternating turns. The UAV would almost never fly level to the ground, and all turns were over $10°$. Circling was also not consistent; the controllers frequently changed direction while within in-range boundary of the radar, rather than orbiting in a circle around the target. For this simplest of fitness measures, evolution tended to select very simple bang-bang control, changing the roll angle at every time step using sharp right and left turns, with the single goal of minimizing the AoA. In a comparison, evolved controllers exhibited slightly better performance than a human-designed, rule-based controller. Further comparisons were not made, because the human-designed controller's performance degraded rapidly as additional fitness measures and radar types were considered.

Using only two fitness measures was also not sufficient to achieve the desired behaviors. If $fitness_1$ and $fitness_2$ were used, the circling behavior improved, but the efficiency of the flight path was unchanged. If $fitness_1$ and $fitness_3$ were used, the UAV would fly level a large amount of the time, but circling was very poor, with larger radius orbits or erratic behavior close to the target. Sharp turns were also very common. If $fitness_1$ and $fitness_4$ were used, turns were shallower, but the UAV still failed to fly with its wings level to the ground for long periods. Circling around the target also became more erratic and the size of the orbits increased.

If three of the fitness measures were used, evolved behavior was improved, but not enough to satisfy the mission goals. If all fitness measures were used except $fitness_2$, the UAV would fly efficiently to the target, staying level and using only shallow turns. Once in range of the radar, circling was generally poor. Evolved controllers either displayed large, circular orbits or very erratic behavior that was unable to keep the UAV close to the radar. If $fitness_1$, $fitness_2$, and $fitness_4$ were used, the UAV would circle well once it flew in range of the radar. While flying toward the radar, the UAV failed to fly level, though turns tended to be shallow. The best combination of three fitness measures was when only $fitness_4$ was removed. In this case, circling was good and the UAV tended to fly straight to the target. The level time fitness measure also tended to keep the turns shallow and to eliminate alternating between right and left turns. However, turn cost was still high, as many turns were sharp.

When all four of the fitness functions were used, the evolved controllers were sufficiently robust. A variety of strategies were evolved, and many controllers were sufficiently fit to be considered successful. The evolved controllers were able to overcome a noisy environment and inaccurate sensor data in tracking and orbiting a radar site. The four fitness measures selected all had an impact on the behavior of the evolved controllers, and all four were necessary to achieve the desired flight characteristics.

5.2 Metrics for Post-evolution Controller Evaluation

Multi-objective optimization is an effective method for evolving solutions to problems with multiple, competing objectives. One problem with the use of multi-objective optimization is the difficulty in comparing results. In traditional evolution, where each individual has a single fitness value, two evolutionary runs can be compared using the best individual from each run. Since multi-objective optimization produces a Pareto front of solutions, rather than a single best solution, the performance of evolution cannot be easily gauged. In order to measure the performance of evolution in this research, a method of evaluating an evolved multi-objective population was needed.

The four fitness functions used in this work represented ways of measuring four different be-

Table 5.1: Baseline values used to measure the performance of evolution.

Fitness Function	Baseline Value
Normalized distance ($fitness_1$)	0.15
Circling distance ($fitness_2$)	4
Level time ($fitness_3$)	1000
Turn cost ($fitness_4$)	0.05

haviors that a UAV controller should exhibit. The performance of evolved controllers on these four behaviors varied, but it was possible to identify certain levels of fitness that could be deemed acceptable. To measure the performance of evolution, a set of baseline values for the four fitness metrics were chosen. These values were those considered minimally successful for the four behaviors desired from a UAV controller. Each controller that satisfied these values was able to successfully complete the task laid out in this research.

The baseline values used in this research are shown in Table 5.1. These values define a minimally successful UAV controller as able to move quickly to the target radar site, circle at an average distance under 2 nmi, fly with the wings level to the ground for at least 1,000 seconds, and turn sharply less than 0.5 percent of the total flight time. If a controller had a normalized distance fitness value ($fitness_1$) of less than 0.15, a circling distance ($fitness_2$) of less than 4 (the circling distance fitness metric squares the distance), a level time ($fitness_3$) of greater than 1,000, and a turn cost ($fitness_4$) of less than 0.05, the evolution was considered successful.

These baseline values were determined largely by observation of early evolved solutions. The baseline value for the circling distance ($fitness_2$) was the easiest to select, because the fitness function has a very direct correspondence with an easily identifiable quantity, the circling radius. Values for the other three fitness functions were more difficult to select, and were chosen based on the results of evolved solutions. The selection of any cutoff value for a particular fitness function is, in many ways, arbitrary. If this human bias were part of the evolutionary process, this might be a cause for concern. However, these baselines values were used only for the analysis of evolved populations, not during the evolutionary process itself. This is an important distinction, because multi-objective evolution is allowed to evolve solutions across the entire Pareto front, not just the segment we as humans would prefer to choose from. This freedom of evolution is important, because individuals that are not necessarily attractive for all four fitness functions contribute to the diversity of the population. These individuals might also contain subtrees that could be used to create better individuals. This method of evaluating the performance of evolution is not only designed to effectively analyze the quality of evolved populations but to avoid adding human bias to the evolutionary process.

5.3 Direct Evolution

This first set of experiments all used direct evolution. Each evolution started with a random initial population and was evolved for 600 generations. Five radar types were used for these experiments: 1) continuously emitting, stationary radars, 2) intermittently emitting, stationary radars with regular periods, 3) intermittently emitting, stationary radars with irregular periods, 4) continuously emitting, mobile radars, and 5) intermittently emitting, mobile radars. Evolution was successful in producing controllers that satisfied the baseline values for acceptable controllers, but evolving controllers for the more complex radar types had a much lower success rate.

5.3.1 Continuously Emitting, Stationary Radar

The first experiment evolved controllers on a stationary, continuously emitting radar. Of all the experiments, this was the simplest task presented to evolution. During the four hour simulation time, the radar continuously emits a signal and never moves. The simulation is set up as described in Chapter 3, and the evolutionary process is performed as described in Chapter 4.

Table 5.2: Results for experiments with continuously emitting, stationary radars.

Number of evolutionary runs	50
Number of successful runs	45
Success percentage	90%
Total number of successful controllers	3,149
Average successful controllers per run	62.98
Maximum successful controllers for a run	170
Minimum successful controllers for a run	1

In this experiment, 50 complete evolutions, or evolutionary runs, were performed. For the evolutionary runs, each new population of 500 individuals was randomly initialized and evolved for 600 generations. Results from this experiment are summarized in Table 5.2. Of the 50 evolutionary runs, 45 runs were acceptable under the baseline values, a 90 percent success rate. The number of acceptable controllers evolved during an individual run ranged from 1 to 170. Figure 5.1 shows a histogram of the number of acceptable controllers evolved in each successful run. This plot shows that the evolutionary runs were in a variety of stages when evolution ended. Many had produced only a few very competent individuals which had yet to spread themselves across the population, as seen by the large spike near zero. However, many of the runs produced at least 50 successful controllers, suggesting that the subtrees leading to successful behavior were being propagated across the population. Overall, 3,149 acceptable controllers were evolved, for an average of 62.98 successful controllers per evolutionary run.

Figure 5.2 shows five sample flight paths to five different emitter locations for an evolved controller. The fitness values for these five flights are shown in Table 5.3. From these five flights, one can see that the evolved controller flies to the target very efficiently, staying level a majority of the time. Almost all turns are shallow. Once in range of the target, the roll angle is gradually increased. Once the roll angle reaches its maximum value to minimize the circling radius, no change to the roll angle is made for the remainder of the simulation. Populations tended to evolve to favor turning left or right.

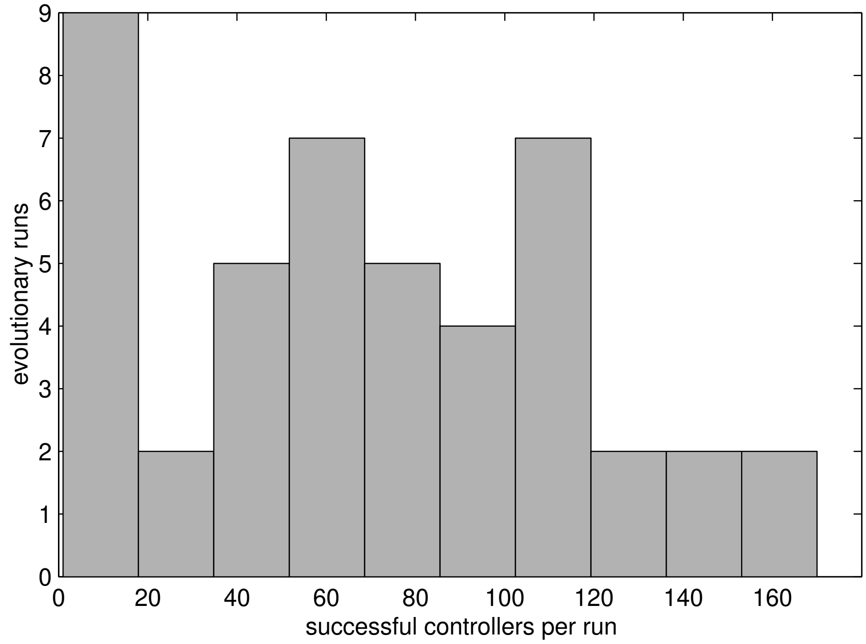


Figure 5.1: Histogram of the number of successful controllers for each evolutionary run for continuously emitting, stationary radars.

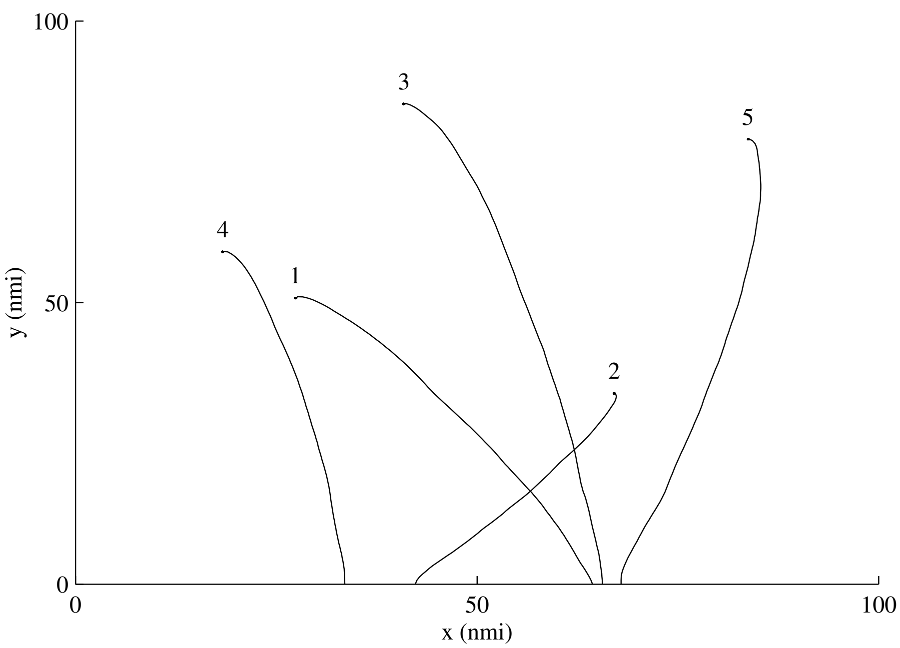


Figure 5.2: Five sample flight paths for an evolved controller flying a UAV to continuously emitting, stationary radars.

Table 5.3: Fitness values for five UAV flight paths to continuously emitting, stationary radars shown in Figure 5.2.

Flight	Normalized Distance	Circling Distance	Level Time	Turn Cost
1	0.067	1.299	2,346	0.014
2	0.044	1.189	1,384	0.007
3	0.094	1.440	3,531	0.023
4	0.064	1.291	2,245	0.014
5	0.085	1.383	3,122	0.008
Baseline	0.15	4	1,000	0.05

Table 5.4: Fitness values for the flight path examined in Figures 5.3, 5.4, and 5.5.

Fitness Function	Fitness Value
Normalized Distance	0.0596
Circling Distance	1.2485
Level Time	2,052
Turn Cost	0.0067

Figures 5.3, 5.4, and 5.5 show the circling behavior of this evolved controller in more detail. The full flight path of the UAV is shown in Figure 5.3 and the fitness values are shown in Table 5.4. Figure 5.4 shows the UAV as it begins to circle around the radar. The UAV approaches the radar flying straight and level, and as it begins to close in, it gradually increases its roll angle by increments of $8°$ until the roll angle reaches the maximum value of $45°$, making corrections when necessary to keep the radar inside the circle. In Figure 5.4, the UAV is just about to complete its first orbit around the radar. In Figure 5.5, the UAV has been orbiting around the radar for several minutes. The UAV is circling in the tightest orbit possible. The radar is not exactly in the center of this circle, but it is very close.

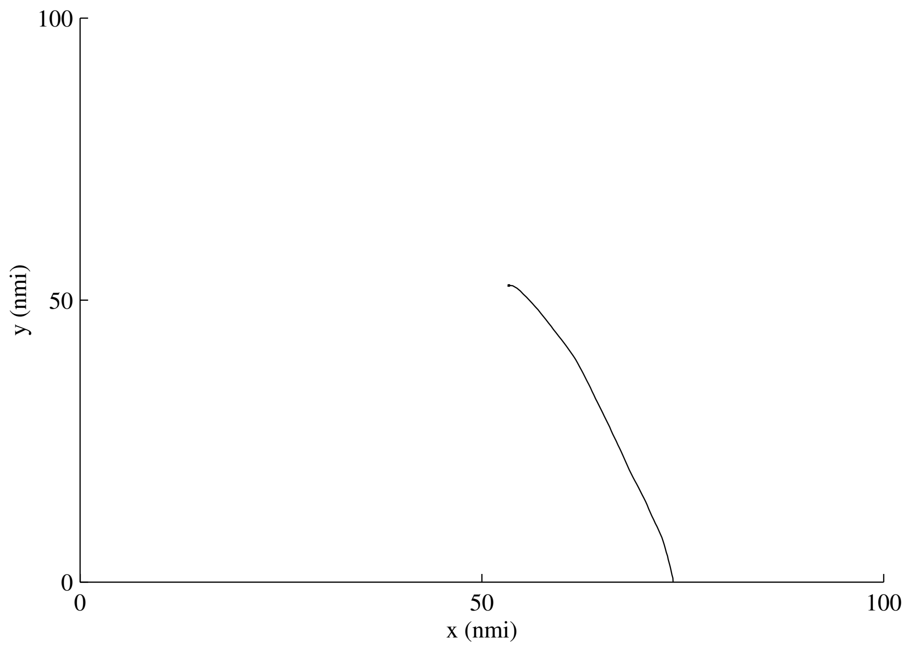


Figure 5.3: A sample flight path for a UAV guided by an evolved controller flying to a continuously emitting, stationary radar.

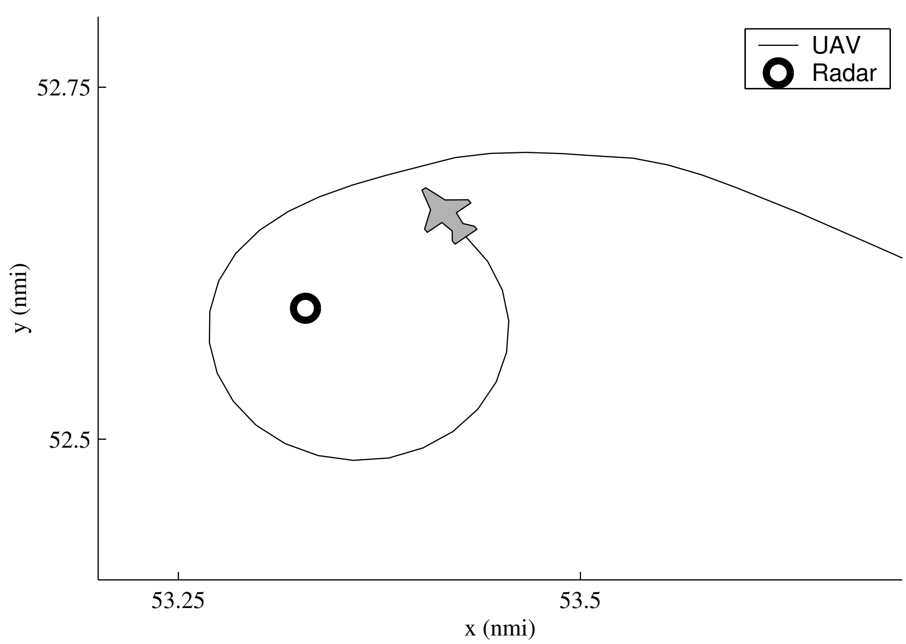


Figure 5.4: A closeup of the UAV flight path shown in Figure 5.3 after 43 minutes and 20 seconds. The UAV has just begun to circle around the radar.

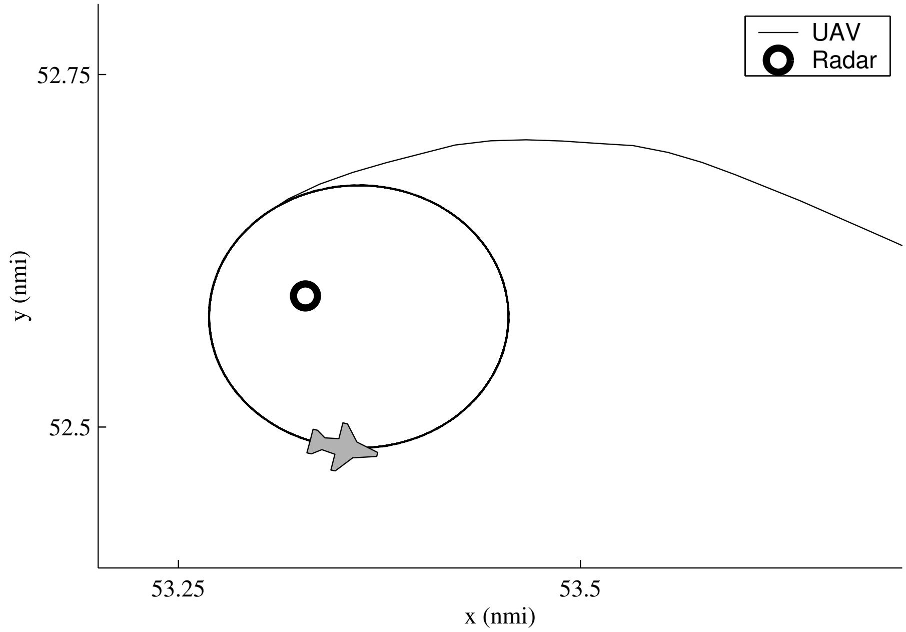


Figure 5.5: A closeup of the UAV flight path shown in Figure 5.3 after 47 minutes and 5 seconds. The UAV is circling around the radar.

5.3.2 Intermittently Emitting, Stationary Radar with Regular Period

The next type of radar examined was an intermittently emitting, stationary radar. Most aspects of the simulation were identical to the continuously emitting case. The initial position of the radar is randomly determined for each simulation trial and the radar is stationary for the entire simulated period. The difference in this radar type is that the emitter is not emitting continuously. Instead, the radar emits periodically. The radar was set to emit for 5 minutes and then turned off for 5 minutes, giving a regular period of 10 minutes and a 50 percent duty cycle. This intermittent rather than continuous emission was the only change from the continuously emitting, stationary case described previously.

Unlike the continuously emitting case, this intermittently emitting radar was quite difficult for evolution. This should come as no surprise; the sensors on-board the UAV receive only half as much information from this type of radar as from a continuously emitting radar. Since the

Table 5.5: Results for experiments with intermittently emitting, stationary radars with regular periods.

Number of evolutionary runs	50
Number of successful runs	25
Success percentage	50%
Total number of successful controllers	1,891
Average successful controllers per run	37.82
Maximum successful controllers for a run	156
Minimum successful controllers for a run	2

controllers evolved in this research have no *a priori* knowledge of the radar's location and no internal model of the world, evolution must devise a strategy for the times when the emitter is turned off.

As in the previous experiment, 50 evolution runs were performed. Again, for each run, a new population was randomly initialized and evolved for 600 generations. The results of this series of evolutionary runs are summarized in Table 5.5. Of the 50 runs, only 25 runs, or 50 percent, were successful. The number of acceptable controllers evolved in successful runs ranged from 2 to 156. Figure 5.6 shows a histogram of the the number of acceptable controllers evolved for each of the 25 successful runs. In comparison to the continuously emitting, stationary radar experiment, the results seen here are clumped more toward zero. While good individuals did manage to spread across some populations, as seen by the large spike near 140, many evolutionary runs were only able to produce a few good controllers. A total of 1,891 successful controllers were evolved for an average of 37.82 acceptable controllers per evolutionary run.

Figures 5.7a, 5.8a, 5.9a, 5.10a, and 5.11a show five flight paths for UAVs flying to intermittently emitting, stationary radars. Figures 5.7b, 5.8b, 5.9b, 5.10b, and 5.11b show the distance between the UAV and the radar for each of these flights during the simulated flight time as well as the emitting durations of the radars. The fitness values for these flights are shown in Table 5.6. The flight paths for these controllers were similar to those for the continuously emitting

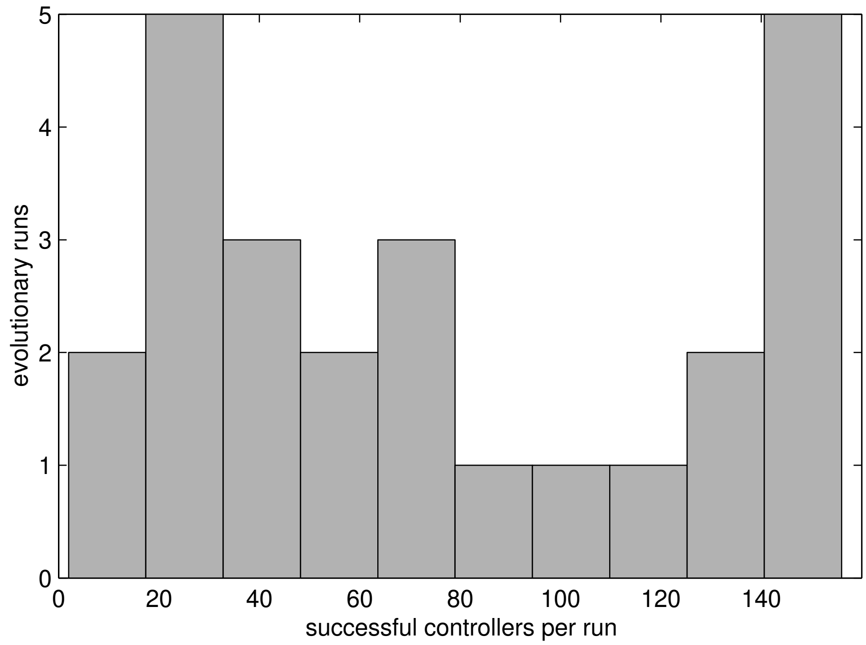
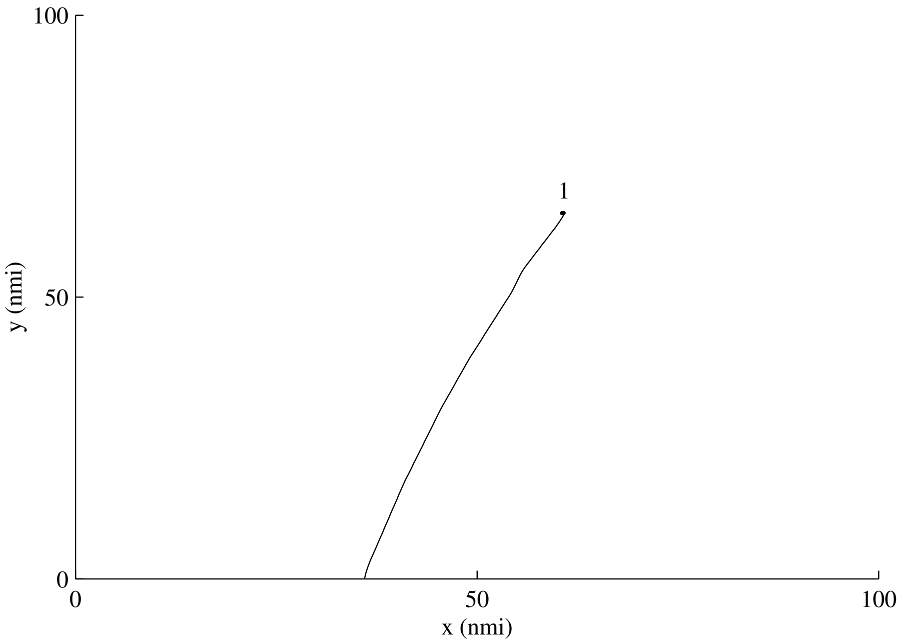


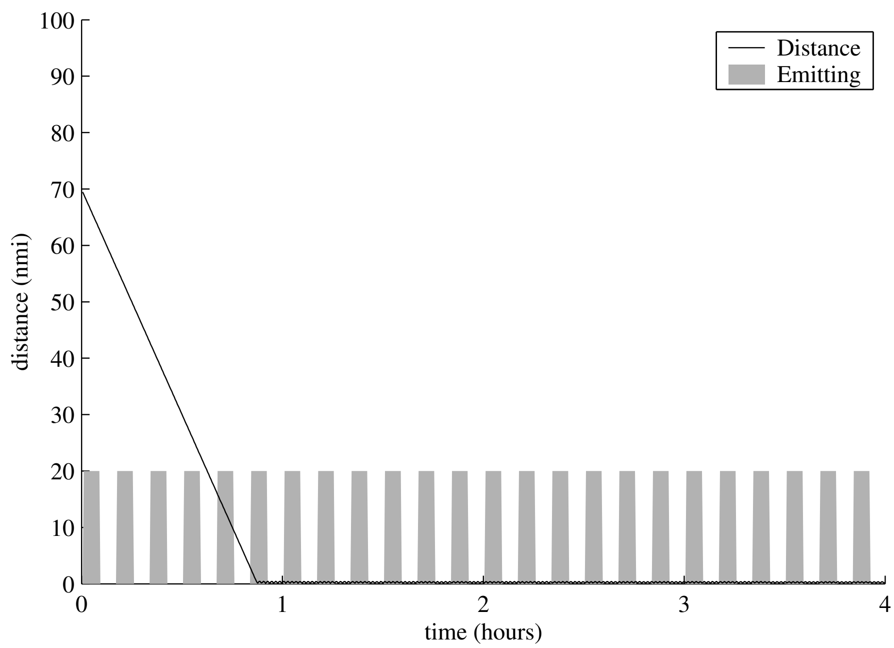
Figure 5.6: Histogram of the number of successful controllers for each evolutionary run for intermittently emitting, stationary radars.

Table 5.6: Fitness values for five UAV flight paths to intermittently emitting, stationary radars shown in Figures 5.7, 5.8, 5.9, 5.10, and 5.11.

Flight	Normalized Distance	Circling Distance	Level Time	Turn Cost
1	0.073	1.363	2,657	0.024
2	0.056	1.333	1,957	0.032
3	0.097	1.505	3,748	0.043
4	0.095	1.422	3,426	0.014
5	0.111	1.505	4,286	0.028
Baseline	0.15	4	1,000	0.05

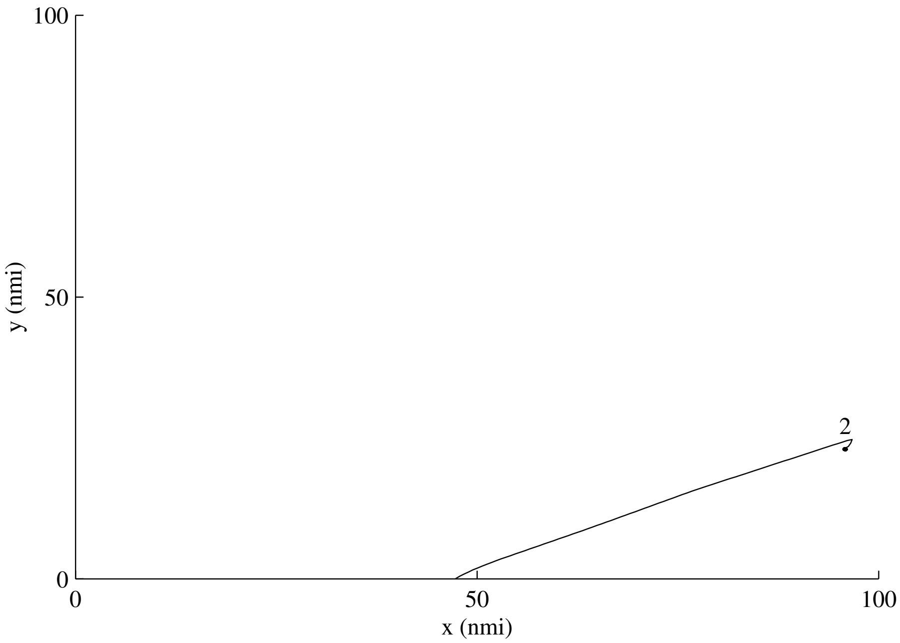


(a)

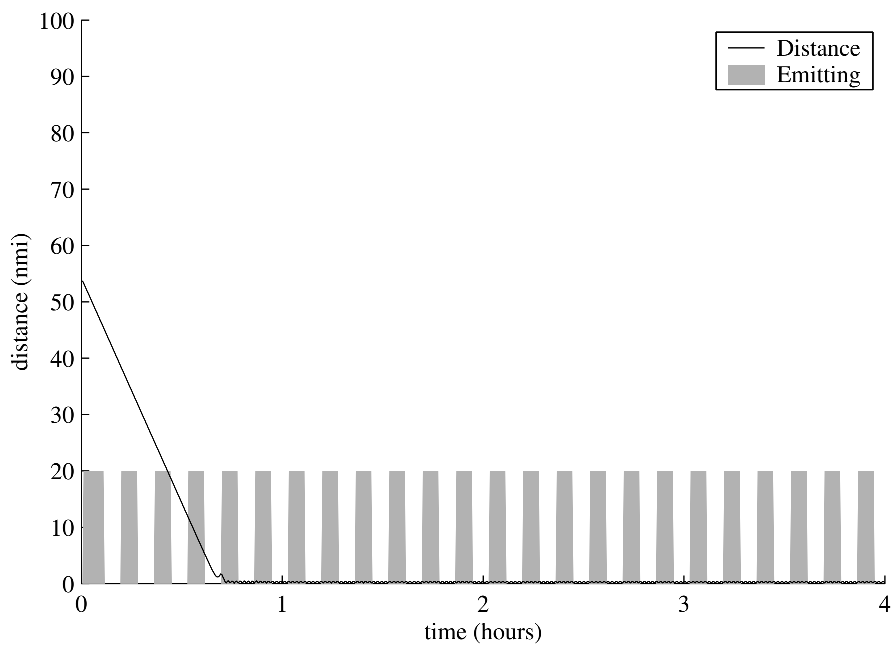


(b)

Figure 5.7: Flight path 1 for a UAV controller to an intermittently emitting, stationary radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).



(a)



(b)

Figure 5.8: Flight path 2 for a UAV controller to an intermittently emitting, stationary radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

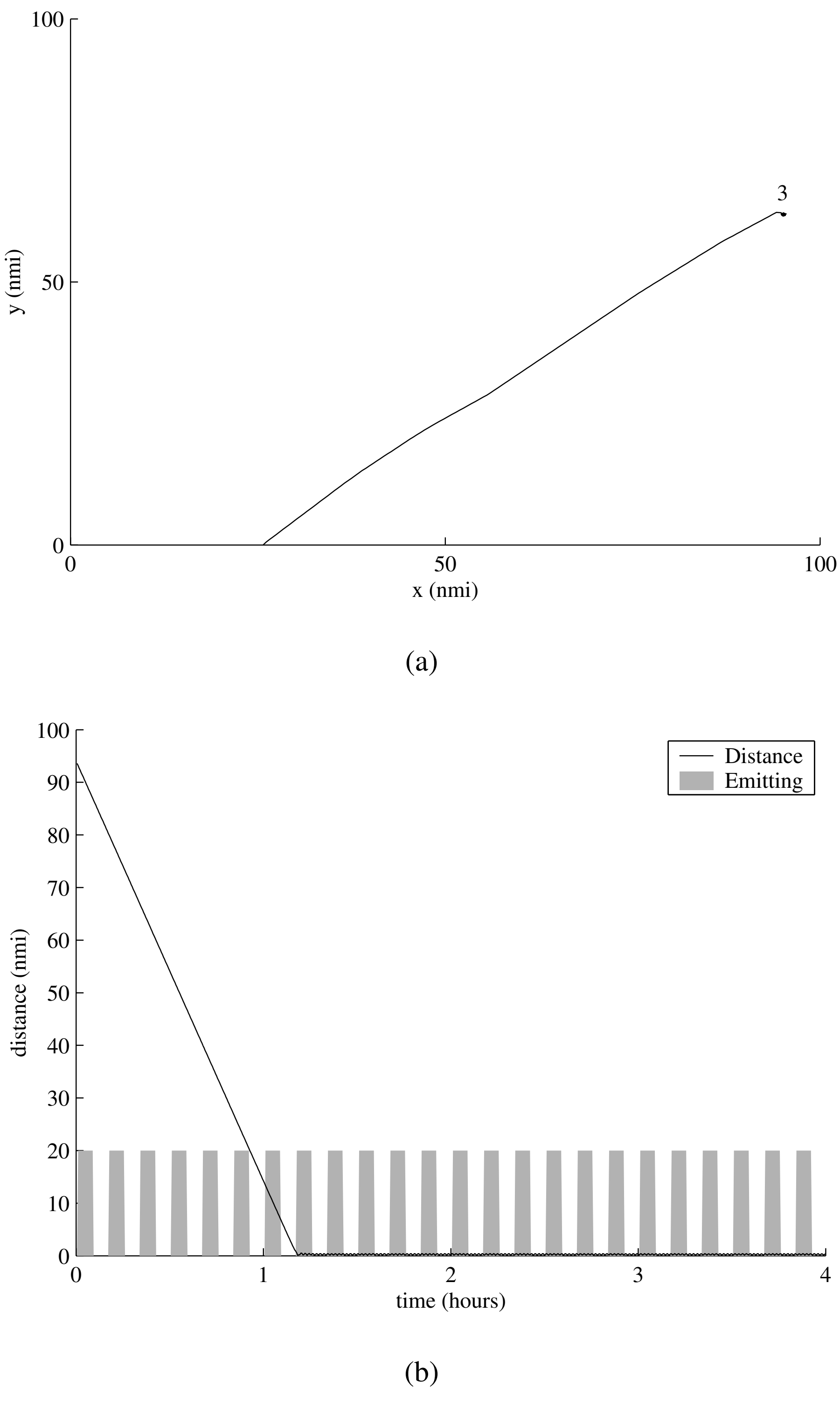
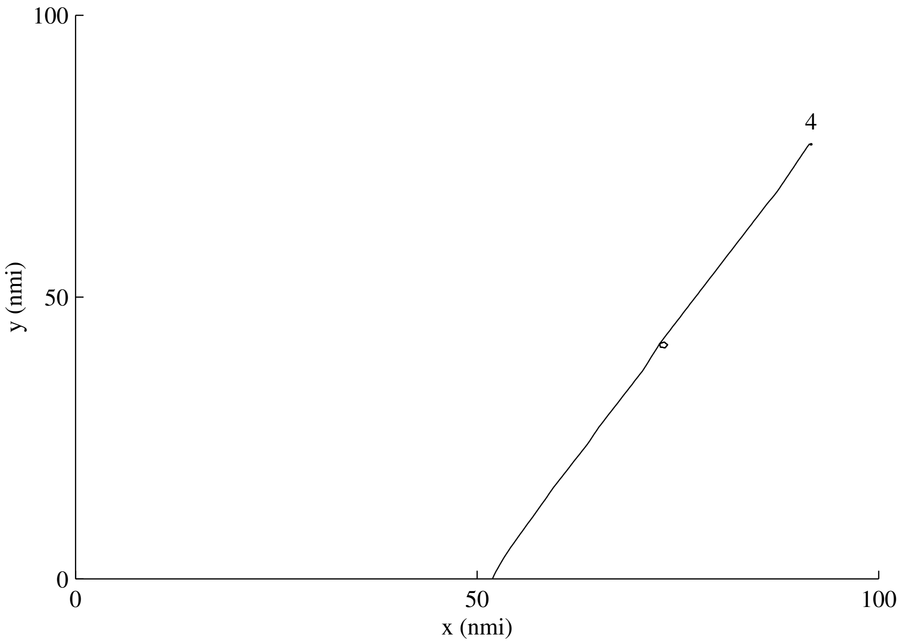
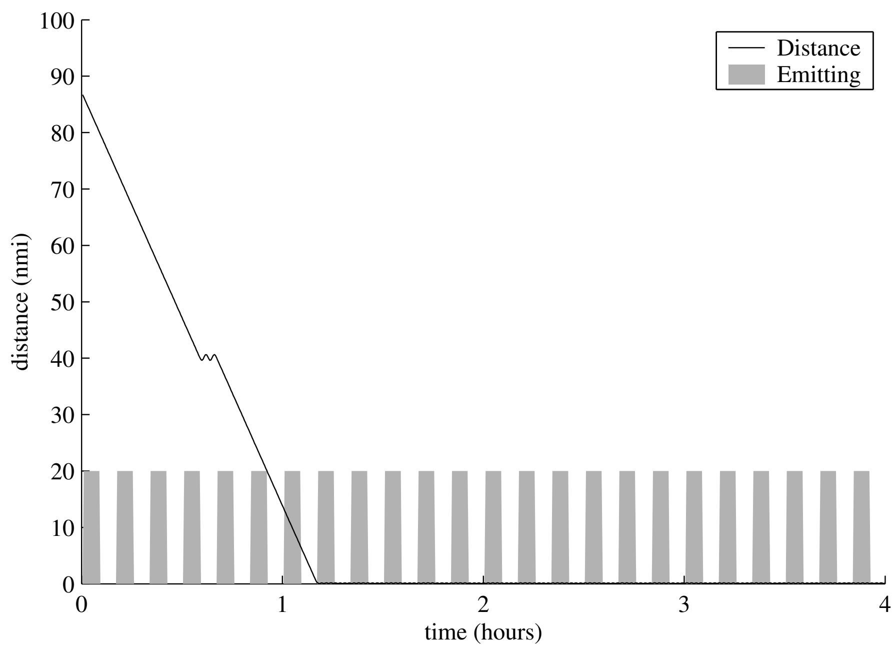


Figure 5.9: Flight path 3 for a UAV controller to an intermittently emitting, stationary radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

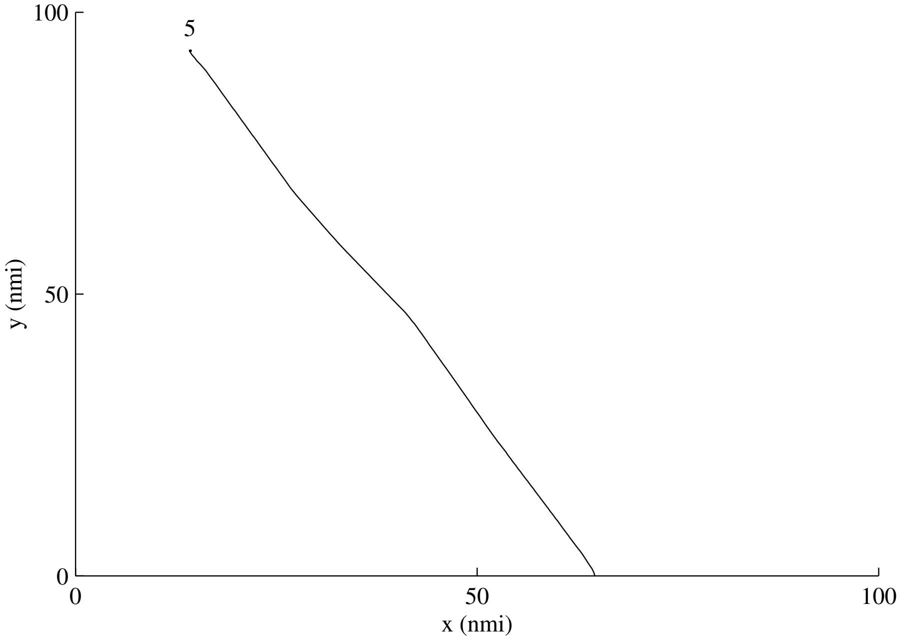


(a)

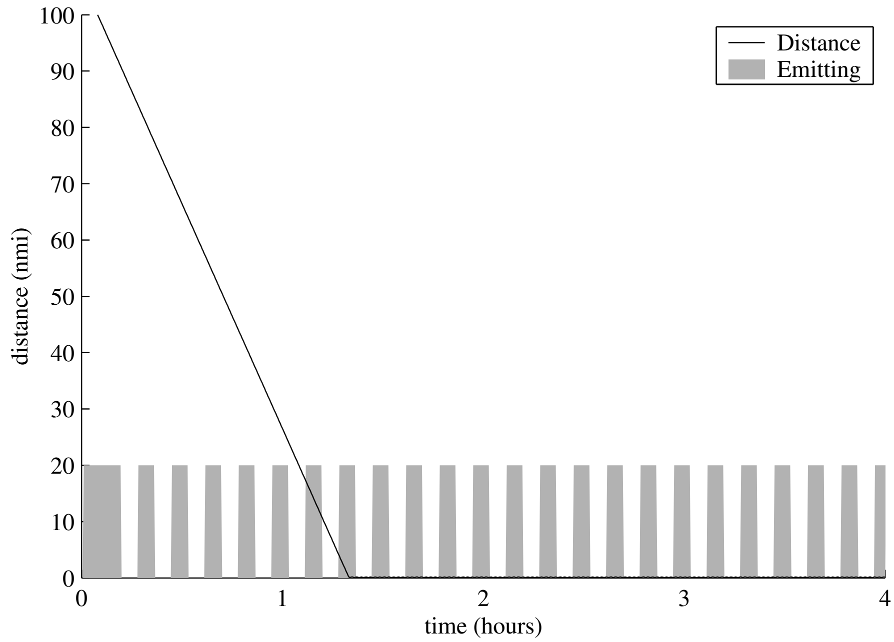


(b)

Figure 5.10: Flight path 4 for a UAV controller to an intermittently emitting, stationary radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).



(a)



(b)

Figure 5.11: Flight path 5 for a UAV controller to an intermittently emitting, stationary radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

radars. In Figures 5.7, 5.9, and 5.11, the UAV flies directly to the radar and then circles around it. In Figure 5.8, the UAV flies directly to the radar, but at the point the UAV reaches the radar's location, the radar is not emitting. The UAV flies past the radar, and once the radar begins to emit again, the UAV returns and begins to circle the radar. Subsequent periods where the radar does not emit do not change the circling behavior of the UAV. In some cases, controllers evolved a waiting behavior, where near the beginning of flight, the UAV would circle during the period when the radar was not emitting. This behavior can be seen in Figure 5.11. At one point in the flight, the radar stops emitting, and until it resumes, the UAV circles approximately 40 nmi from the radar. In some cases, when the UAV is unsure of the proper direction to head, this behavior is useful. For the best controllers this behavior is very infrequent.

Despite the increased difficulty of this experiment, evolution was able to produce a large number of successful controllers. The vast drop in the success rate of evolution can be attributed to the difficulty posed by having sensor information only 50 percent of the time. This makes evolving very good controllers more difficult.

5.3.3 Intermittently Emitting, Stationary Radar with Irregular Period

The next experiment was also done on intermittently emitting, stationary radars. Unlike the previous experiment, the radar used in this series of evolutionary runs did not have a regular period. Both the period and the duration of emission were random within certain bounds. The minimum period was 8 minutes and the mean period was 12 minutes. The minimum emitting duration was 4 minutes and the mean emitting duration was 6 minutes. At every new period, a new period length and duration length were set randomly. Like the intermittently emitting case with a regular emission period, this radar was difficult for evolution. Like the periodically emitting radar, there were frequent periods of no emission, where the UAV was unable to sense its target at all. However, these times were more random in length and frequency.

Table 5.7: Results for experiments with intermittently emitting, stationary radars with irregular periods.

Number of evolutionary runs	50
Number of successful runs	29
Success percentage	58%
Total number of successful controllers	2,374
Average successful controllers per run	47.48
Maximum successful controllers for a run	172
Minimum successful controllers for a run	8

As in the previous experiment, 50 evolution runs were performed. For each run, a new population was randomly initialized and evolved for 600 generations. The results are summarized in Table 5.7. Out of the 50 runs, 29 runs were successful, for a 58 percent success rate. The number of acceptable controllers evolved in successful runs ranged from 8 to 172. Figure 5.12 shows a histogram of the the number of acceptable controllers evolved for each of the 29 successful runs. A total of 2,374 successful controllers were evolved for an average of 47.48 acceptable controllers per evolutionary run.

Figures 5.13a, 5.14a, 5.15a, 5.16a, and 5.17a show five flight paths for UAVs flying to intermittently emitting, stationary radars. Figures 5.13b, 5.14b, 5.15b, 5.16b, and 5.17b show the distance between the UAV and the radar for each of these flights during the simulated flight time as well as the emitting durations of the radars. The fitness values for these flights are shown in Table 5.8. The flight paths for these controllers were similar to those for the intermittently emitting radars with regular periods. In Figure 5.14, the UAV flies directly to the radar and then circles around it. In Figures 5.15, 5.16, and 5.17, the UAV flies directly to the radar, but passes the radar while the radar is not emitting. Once the radar begins to emit again, the UAV returns and begins to circle around the emitter. As for those radars with a regular period, some controllers evolved a waiting behavior, where near the beginning of flight, the UAV would circle during the period when the radar was not emitting. A slightly different form

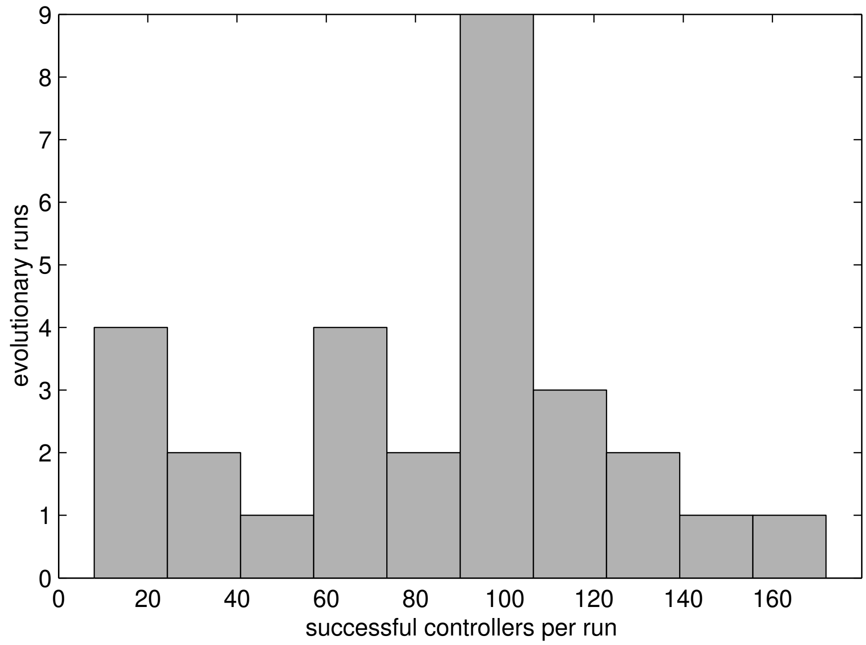
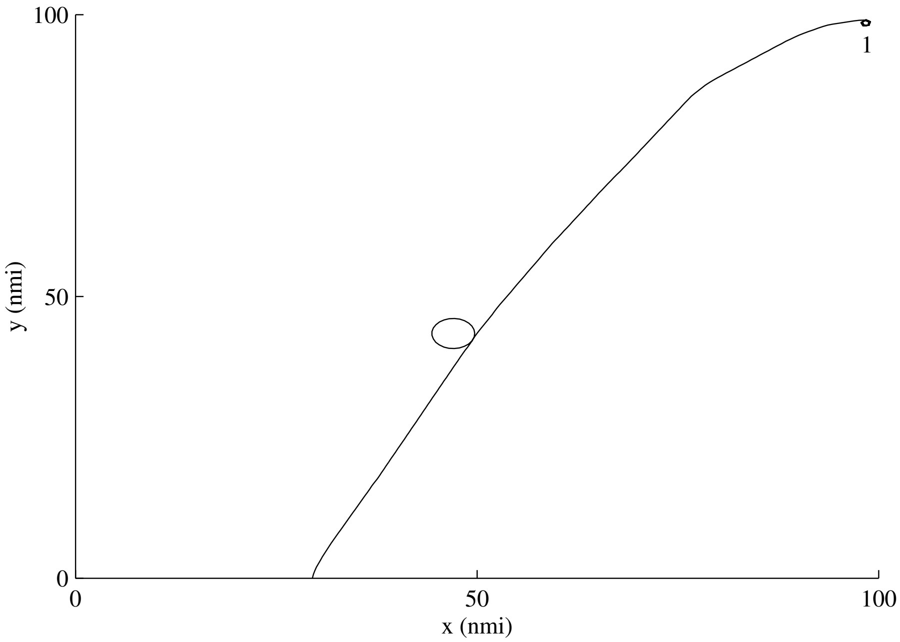


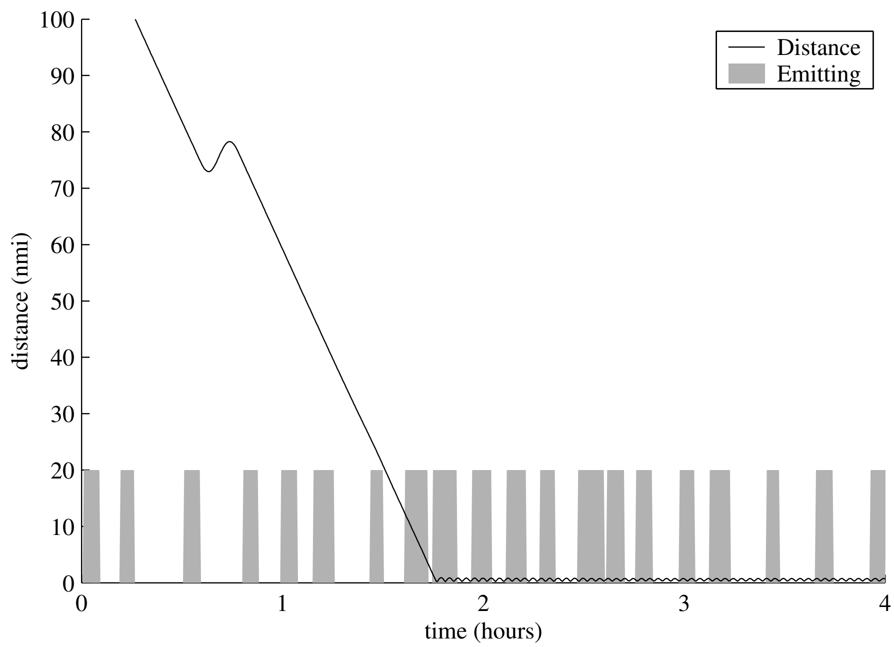
Figure 5.12: Histogram of the number of successful controllers for each evolutionary run for intermittently emitting, stationary radars with an irregular period.

Table 5.8: Fitness values for five UAV flight paths to intermittently emitting, stationary radars with irregular periods shown in Figures 5.13, 5.14, 5.15, 5.16, and 5.17.

Flight	Normalized Distance	Circling Distance	Level Time	Turn Cost
1	0.149	2.148	4,963	0.000
2	0.053	1.593	1,622	0.002
3	0.056	1.821	1,828	0.000
4	0.077	1.697	2,684	0.000
5	0.066	1.740	2,273	0.001
Baseline	0.15	4	1,000	0.05

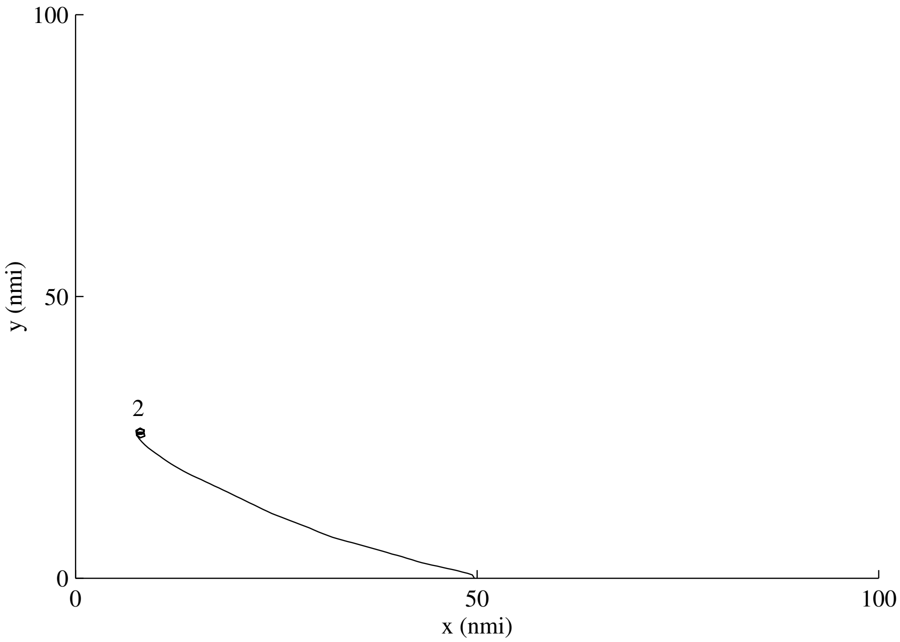


(a)

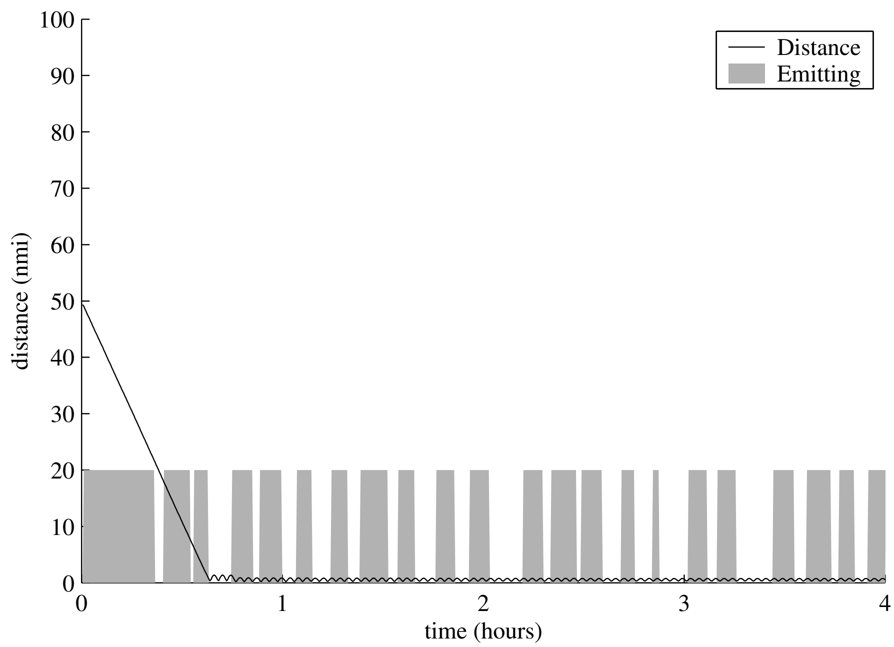


(b)

Figure 5.13: Flight path 1 for a UAV controller to an intermittently emitting, stationary radar with an irregular period (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

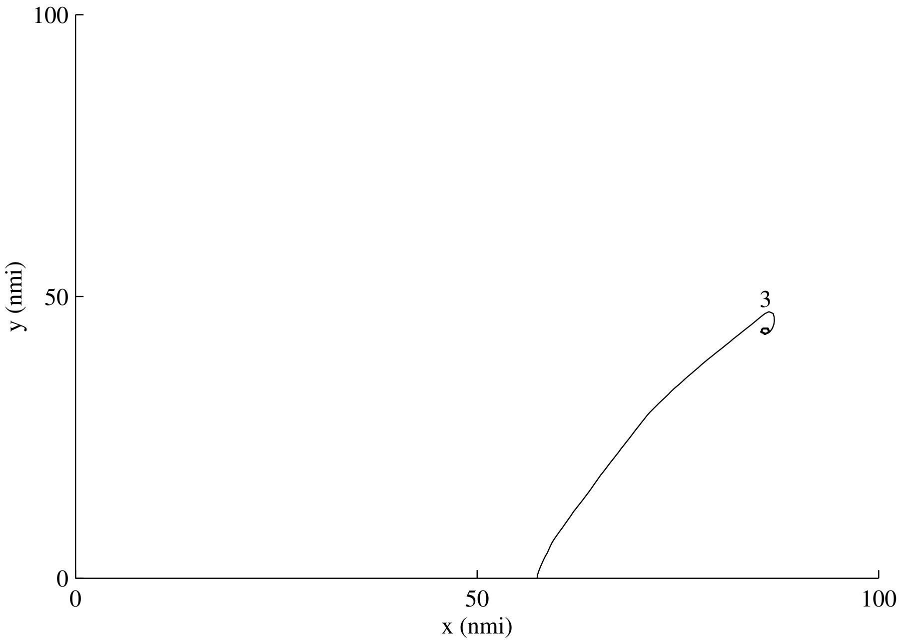


(a)

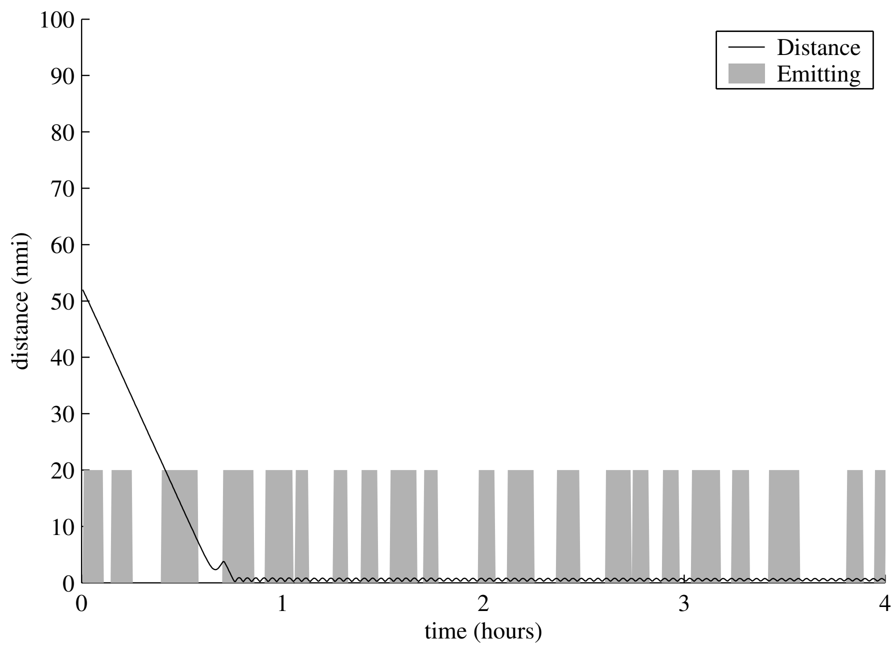


(b)

Figure 5.14: Flight path 2 for a UAV controller to an intermittently emitting, stationary radar with an irregular period (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

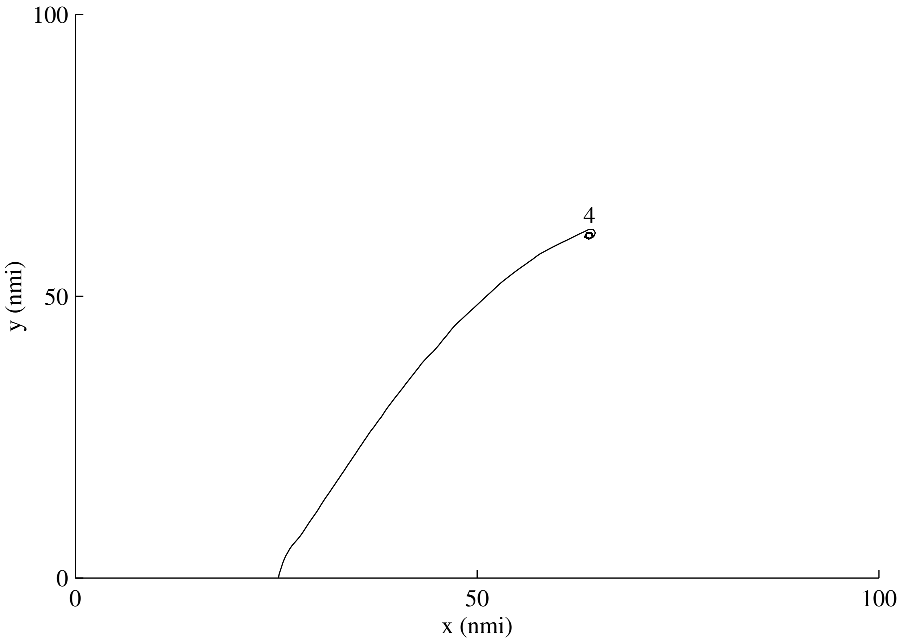


(a)

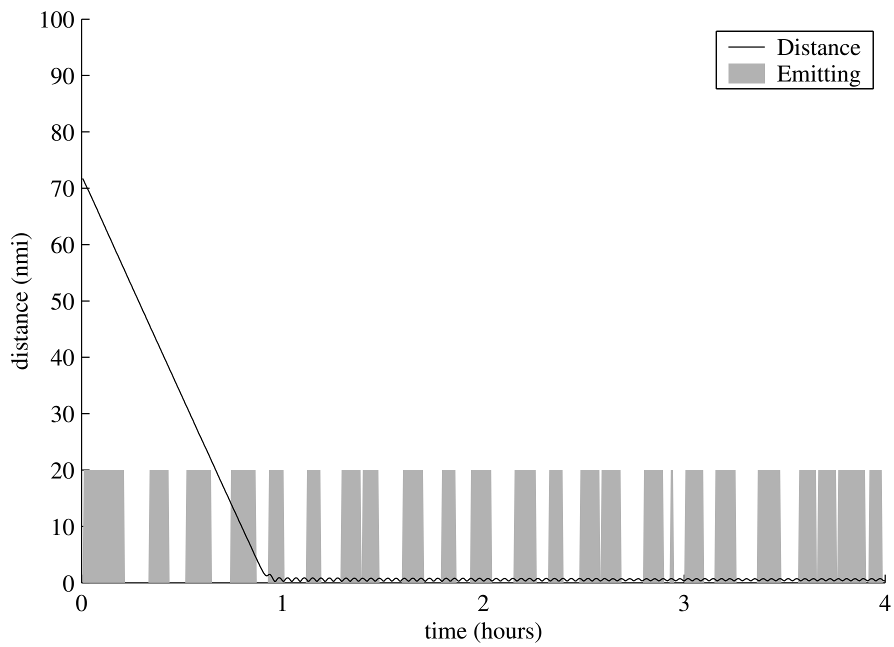


(b)

Figure 5.15: Flight path 3 for a UAV controller to an intermittently emitting, stationary radar with an irregular period (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

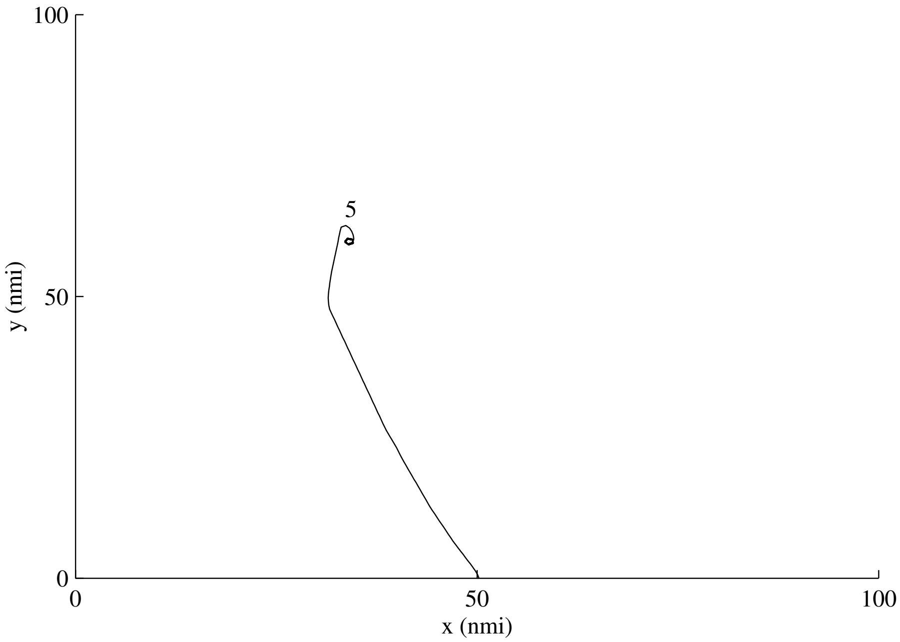


(a)

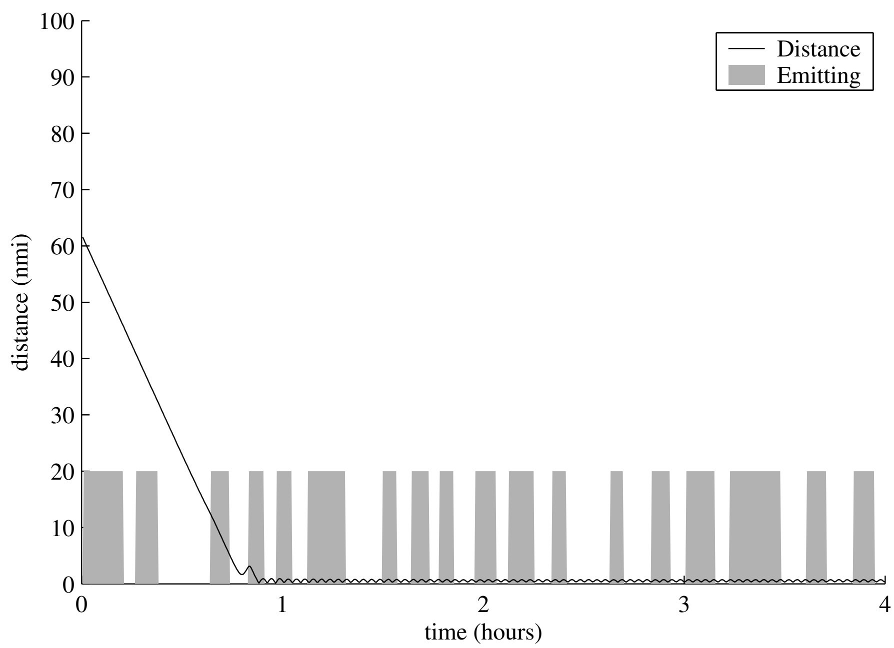


(b)

Figure 5.16: Flight path 4 for a UAV controller to an intermittently emitting, stationary radar with an irregular period (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).



(a)



(b)

Figure 5.17: Flight path 5 for a UAV controller to an intermittently emitting, stationary radar with an irregular period (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

of this behavior than was seen previously can be seen in Figure 5.13. At one point in the flight, the radar stops emitting, and until it resumes, the UAV circles very widely approximately 75 nmi from the radar. Again, in some cases, this behavior is useful. For the best controllers this behavior is very infrequent.

This experiment was slightly easier for evolution than the previous series with a regular period, though the results were similar. In all likelihood, this difference is related to beginning the circling behavior. For radars with a regular period, there are always 5 minutes ranges when they might pass by the radar. While these segments of time might be longer for radars with an irregular period, they might also be shorter. In the 5 flight paths shown, one can identify periods of time where the radar is emitting more than half the time, and this is often when the UAV finds the radar. As shown previously, once the radar has been properly found, the UAV has no difficulty staying nearby, so the difficult part of the problem is finding the radar. For irregular periods of the type used here, this is often easier, making the success of evolution slightly better than for intermittently emitting radars with regular periods.

5.3.4 Continuously Emitting, Mobile Radar

The next type of radar to be explored was a continuously emitting, mobile radar. The mobility was modeled as a finite state machine with the following states: *move*, *setup*, *deployed*, and *tear down*. When the radar moves, it is placed randomly anywhere in the simulation area. The finite state machine is executed for the duration of simulation. The radar site only emits when it is in the *deployed* state; while the radar is in the *move* state, the UAV receives no sensory information. The time in each state is probabilistic, and the minimum amount of time spent in the deployed state is an hour or 25 percent of the simulation time. The simulation is identical to the other experiments other than the configuration of the radar site's movement.

Table 5.9: Results for experiments with continuously emitting, mobile radars.

Number of evolutionary runs	50
Number of successful runs	36
Success percentage	72%
Total number of successful controllers	2,266
Average successful controllers per run	45.32
Maximum successful controllers for a run	206
Minimum successful controllers for a run	1

Like intermittently emitting radars, mobile radars proved more difficult for evolution than continuously emitting, stationary radars. This was true not only because the radar changes location, forcing the UAV to leave the circling state and relocate the radar, but because while the radar is moving, the UAV receives no sensor data from the radar. Like the intermittently emitting radar, there are periods where the radar is invisible to the UAV. The difficulty of this problem also depends highly on the random placements of the radars. The large number of trials used to evaluate an individual in this research are particularly beneficial for evolving controllers for this type of radar.

For 50 evolutionary runs, a new population was randomly initialized and evolved for 600 generations. The results of this series of evolutionary runs are summarized in Table 5.9. Of 50 evolutionary runs, 36 were successful. The number of successful controllers evolved in a run ranged from 1 to 206. Figure 5.18 shows a histogram of the number of acceptable controllers evolved for each of the 36 successful runs. Successful controllers had a hard time spreading throughout the population, evidenced by the large number of runs that produced less than 50 successful controllers. A total of 2,266 successful controllers were evolved for an average of 45.32 successful controllers per evolutionary run.

Figures 5.19a, 5.20a, 5.21a, 5.22a, and 5.23a show five flight paths for UAVs flying to continuously emitting, mobile radars. Figures 5.19b, 5.20b, 5.21b, 5.22b, and 5.23b show the distance

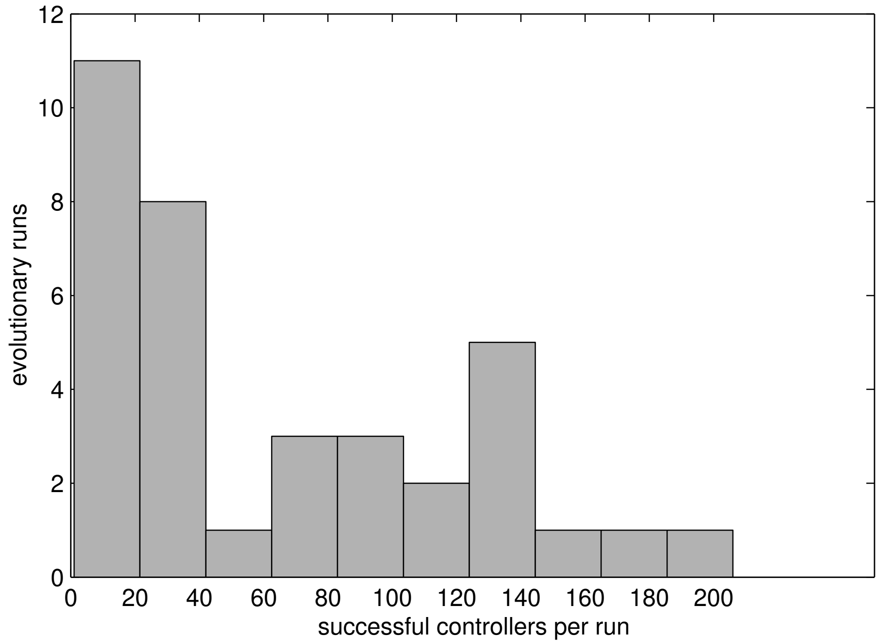
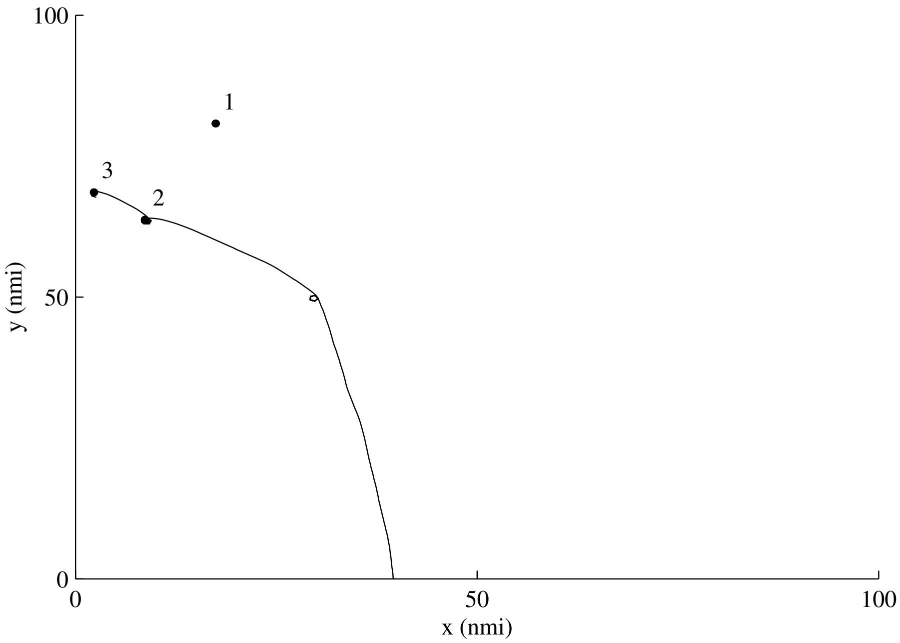
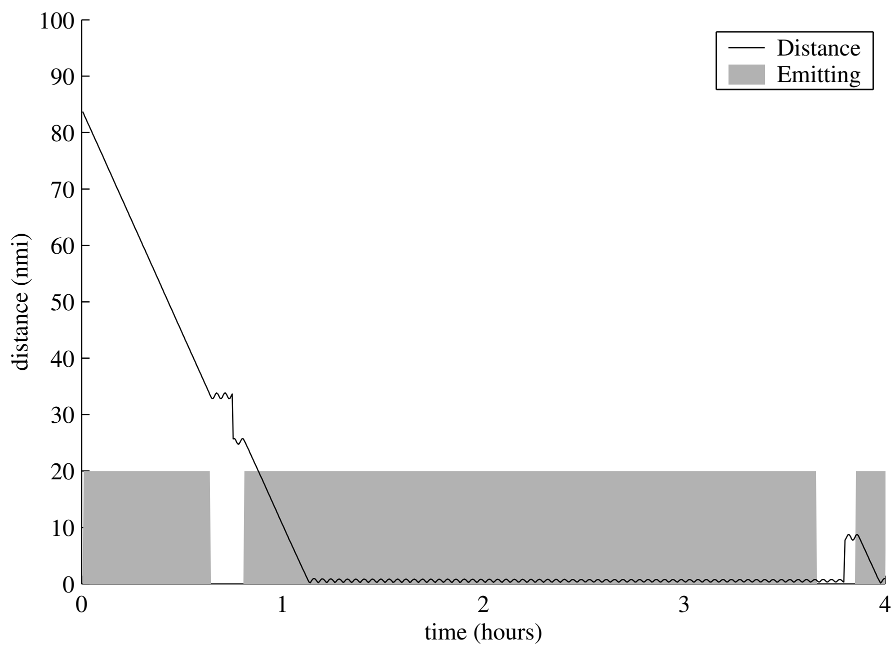


Figure 5.18: Histogram of the number of successful controllers for each evolutionary run for continuously emitting, mobile radars.

between the UAV and the radar for each of these flights during the simulated flight time as well as the emitting durations of the radars. The fitness values for these flights are shown in Table 5.10. In Figure 5.19, the UAV is flying to the radar when the radar stops emitting and then moves. Since the UAV has no idea where to go, it circles in place until it receives new sensory information from the radar. After finding the radar's new location, it circles around the radar until the radar moves again. At the very end of the simulation period, the UAV finds the radar's third location and begins circling around the radar. In Figures 5.20, 5.21, and 5.23, the UAV flies directly to the radar and after the radar has moved, the UAV quickly finds it again. In Figures 5.21 and 5.23, the UAV flies away from the radar briefly because the radar begins emitting again as the UAV is headed away from the radar. In Figure 5.22, the UAV begins to fly to the first position, but he radar moves very quickly with almost no pause in deployment. The UAV then moves very quickly to the second position. The UAV never receives sensor information from the radar at the third position because it never deploys before moving to the

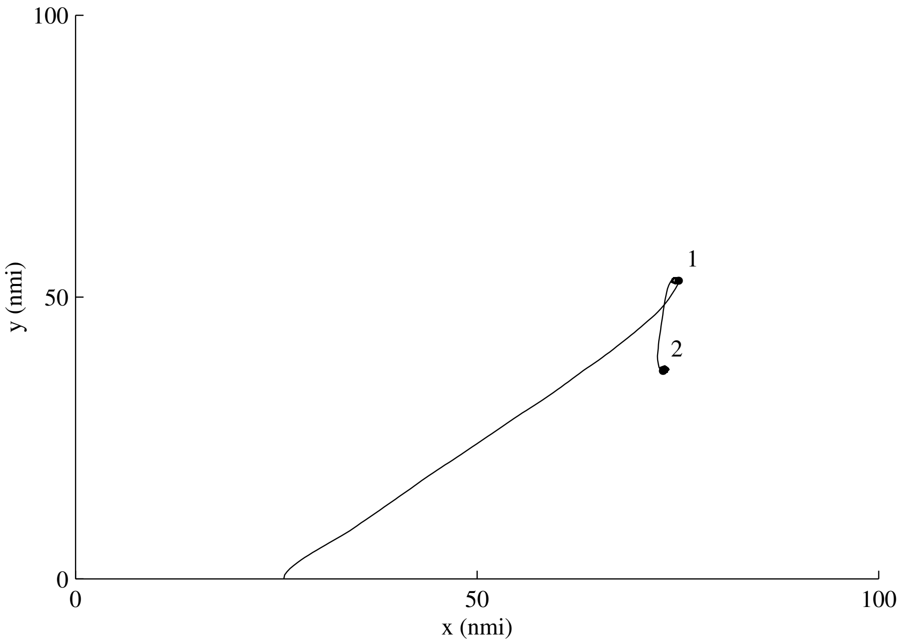


(a)

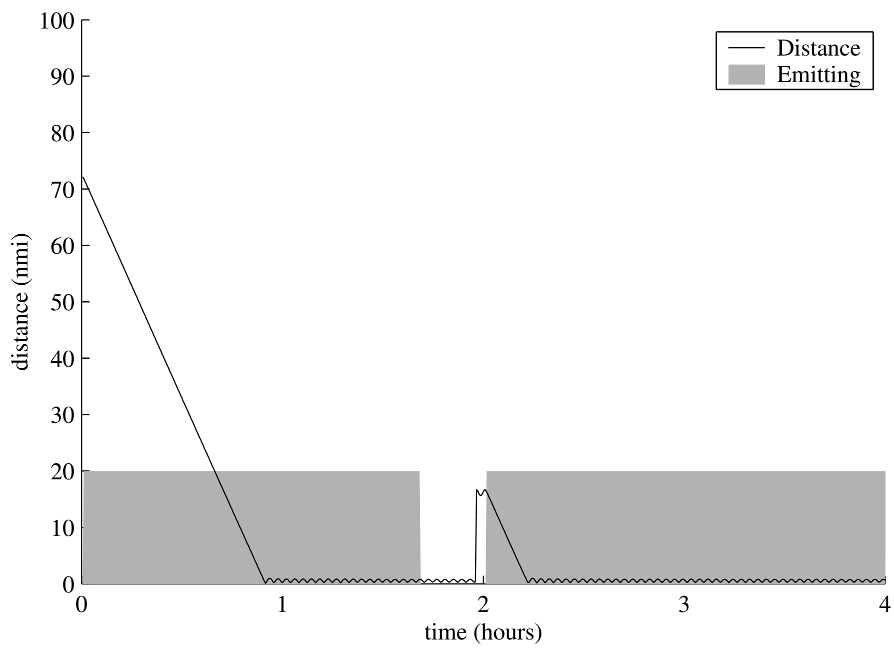


(b)

Figure 5.19: Flight path 1 for a UAV controller to a continuously emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

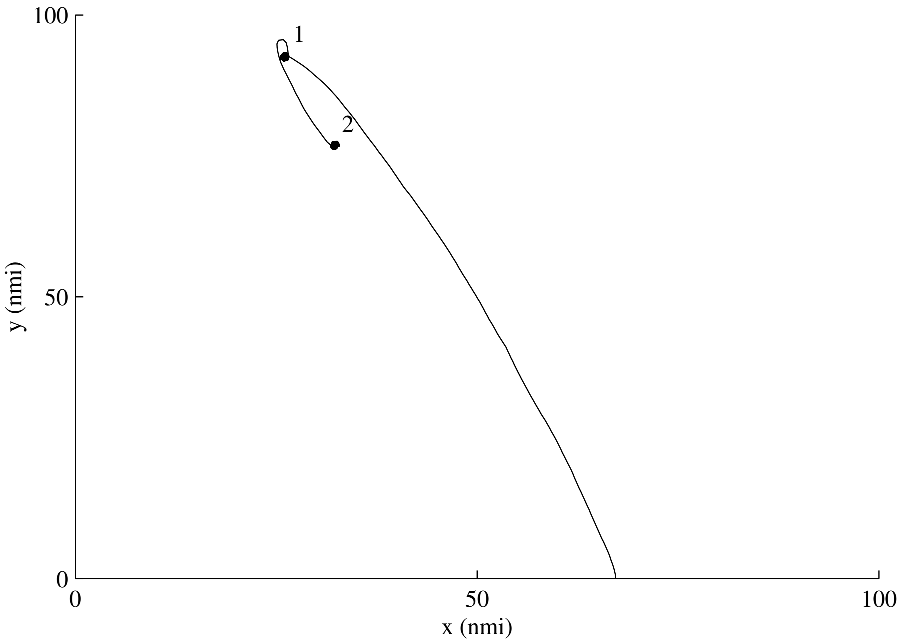


(a)

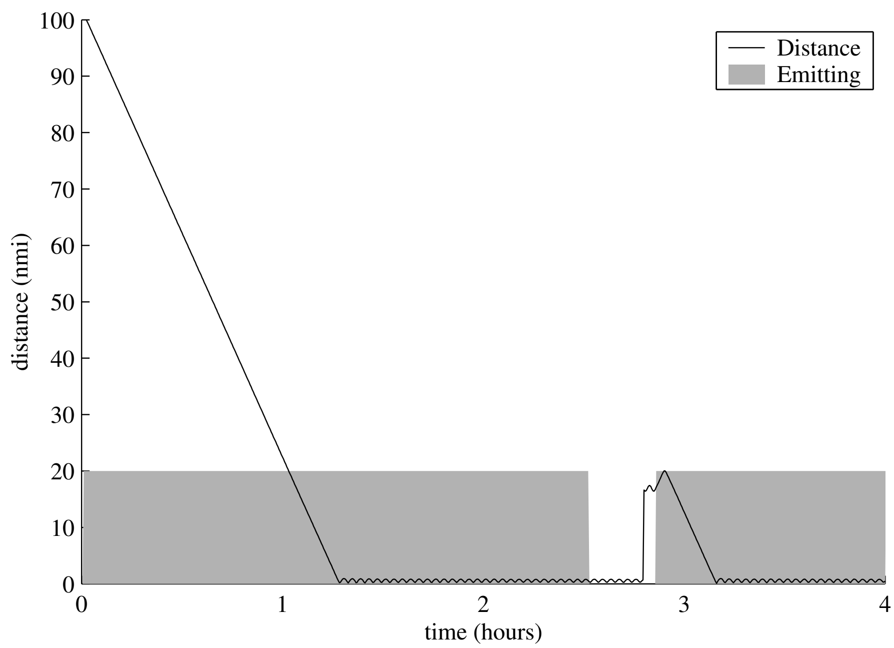


(b)

Figure 5.20: Flight path 2 for a UAV controller to a continuously emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

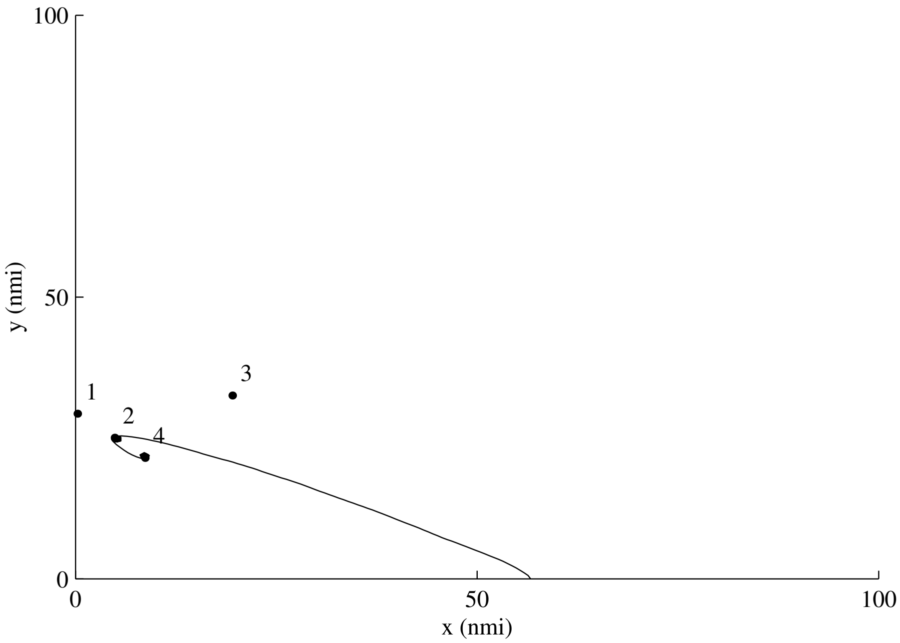


(a)

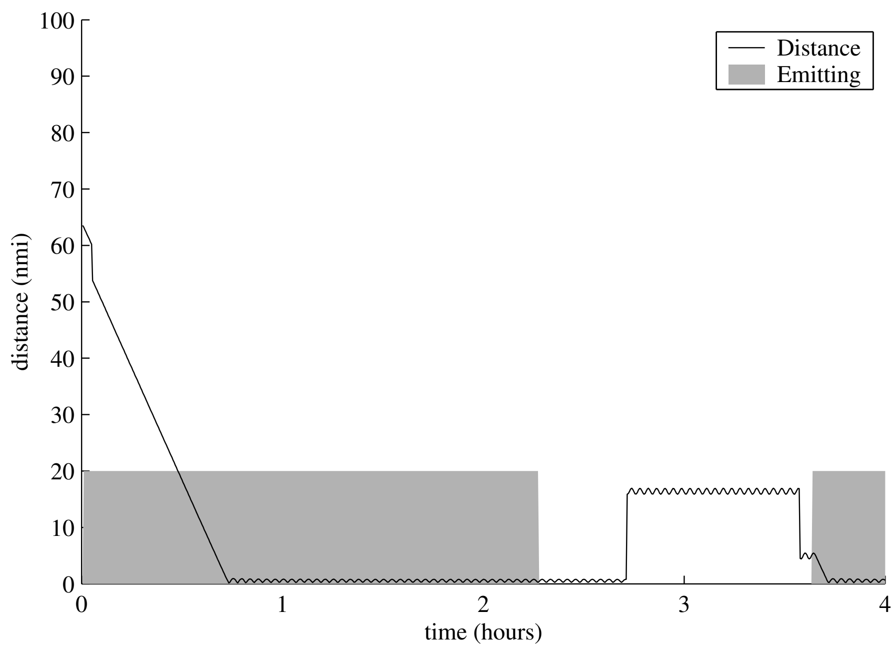


(b)

Figure 5.21: Flight path 3 for a UAV controller to a continuously emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

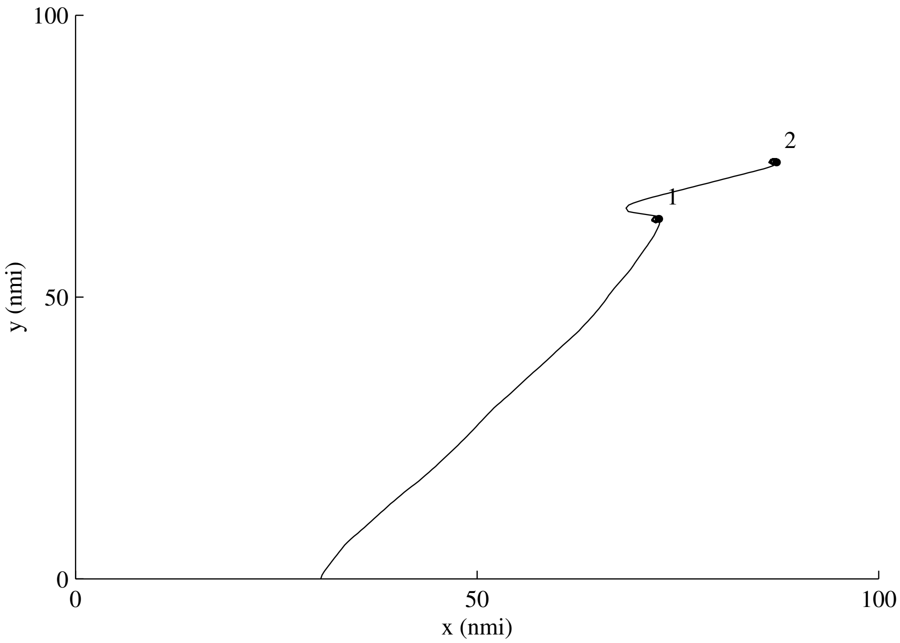


(a)

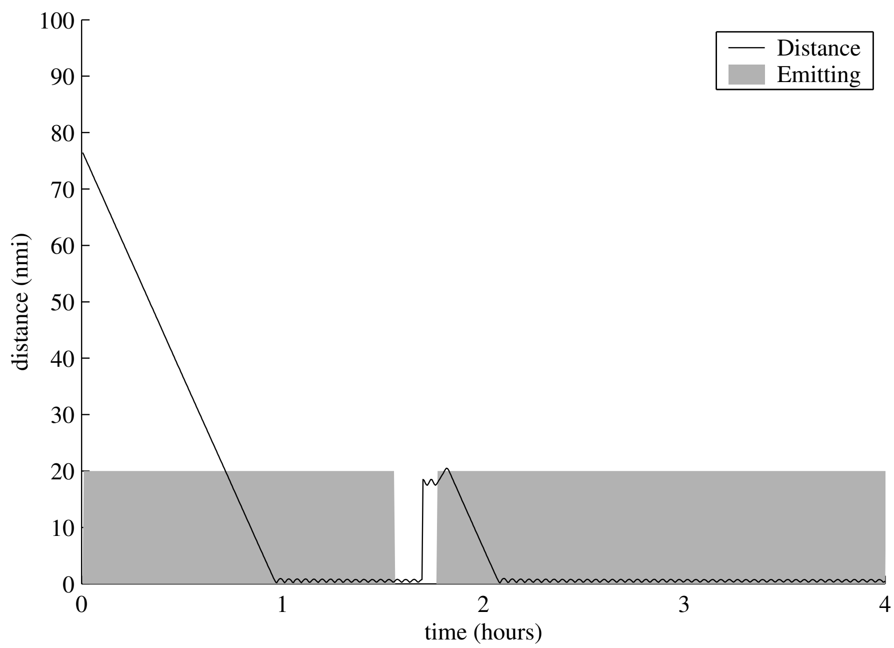


(b)

Figure 5.22: Flight path 4 for a UAV controller to a continuously emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).



(a)



(b)

Figure 5.23: Flight path 5 for a UAV controller to a continuously emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

Table 5.10: Fitness values for five UAV flight paths to continuously emitting, mobile radars shown in Figures 5.19, 5.20, 5.21, 5.22, and 5.23.

Flight	Normalized Distance	Circling Distance	Level Time	Turn Cost
1	0.091	4.279	2,929	0.000
2	0.078	3.079	2,997	0.000
3	0.108	3.568	4,578	0.000
4	0.067	2.969	2,088	0.000
5	0.084	3.211	3,507	0.001
Baseline	0.15	4	1,000	0.05

fourth position. Once the radar has moved to its fourth and final position, the UAV flies to it quickly.

Like the intermittently emitting radar, the mobile radar adds difficulty to the simplest radar case. However, the mobile radar is not as difficult for evolution as the intermittently emitting, stationary radar. This helps to suggest that the movement of the radar is not the most difficult part of the problem. Instead, evolution has the hardest time finding strategies to cope with the periods of sensor blindness.

5.3.5 Intermittently Emitting, Mobile Radar with Regular Period

The final type of radar to be explored with direct evolution was an intermittently emitting, mobile radar. This radar combined the intermittently emitting radar with a regular period with a mobile radar site. The radar emissions were modeled as a combination of the two types of radars. The radar moves using the same finite state machine as the continuously emitting, mobile radar does, but during the deployed state, the radar emits with a regular period. This type of radar was anticipated to be the most difficult for evolution because it combined all the most difficult aspects of the other experiments into a single radar type.

For each of the 50 evolutionary runs, a new population was randomly initialized and evolved

Table 5.11: Results for experiments with intermittently emitting, mobile radars.

Number of evolutionary runs	50
Number of successful runs	16
Success percentage	32%
Total number of successful controllers	569
Average successful controllers per run	11.38
Maximum successful controllers for a run	93
Minimum successful controllers for a run	1

Table 5.12: Fitness values for five UAV flight paths to intermittently emitting, mobile radars shown in Figures 5.25, 5.26, 5.27, 5.28, and 5.29.

Flight	Normalized Distance	Circling Distance	Level Time	Turn Cost
1	0.048	3.187	1,464	0.006
2	0.089	3.527	3,234	0.003
3	0.057	1.954	1,288	0.005
4	0.050	3.168	1,958	0.004
5	0.055	3.132	1,795	0.001
Baseline	0.15	4	1,000	0.05

for 600 generations. The results of this series of evolutionary runs are shown in Table 5.11. Of 50 evolutionary runs, only 16 were successful, for a success percentage of 32 percent. The number of successful controllers evolved in a run ranged from 1 to 93. Figure 5.24 shows a histogram of the number of acceptable controllers evolved for each of the 16 successful runs. A total of 569 successful controllers were evolved for an average of 11.38 successful controllers per evolutionary run.

Figures 5.25a, 5.26a, 5.27a, 5.28a, and 5.29a show five flight paths for UAVs flying to continuously emitting, mobile radars. Figures 5.25b, 5.26b, 5.27b, 5.28b, and 5.29b show the distance between the UAV and the radar for each of these flights during the simulated flight time as well as the emitting durations of the radars. The fitness values for these flights are shown in Table 5.10. In Figure 5.25, the UAV flies directly to the first radar position and after the radar moves

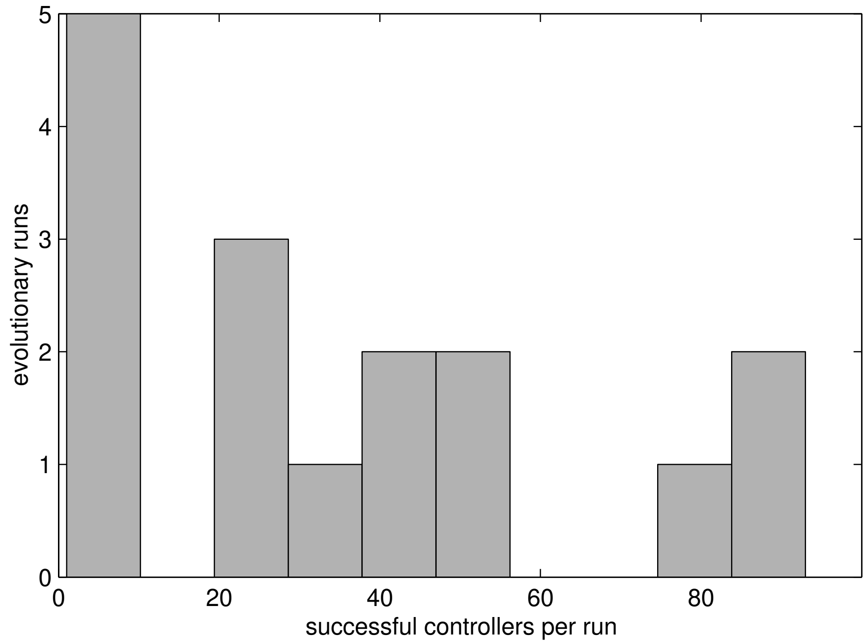
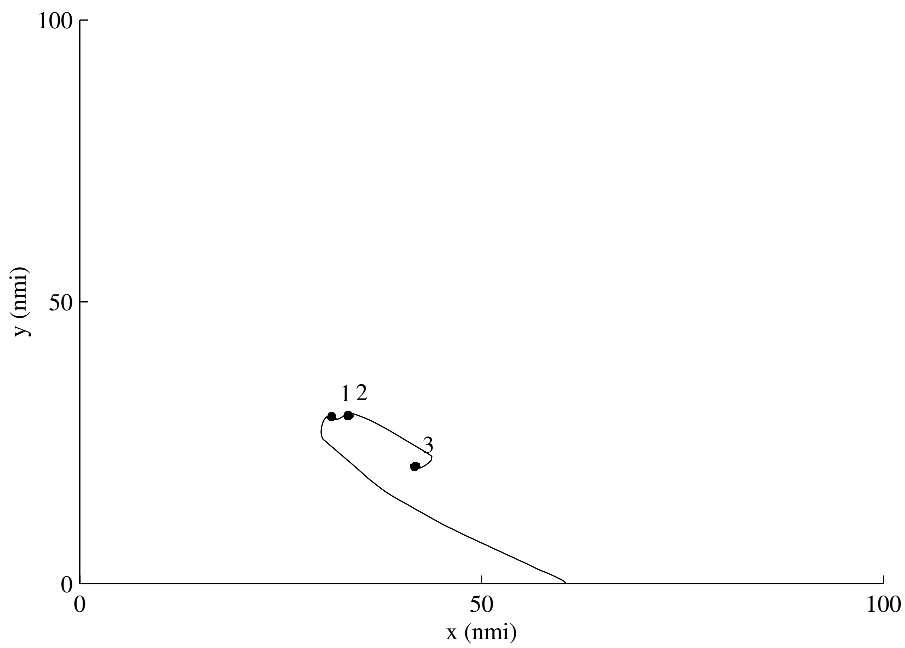


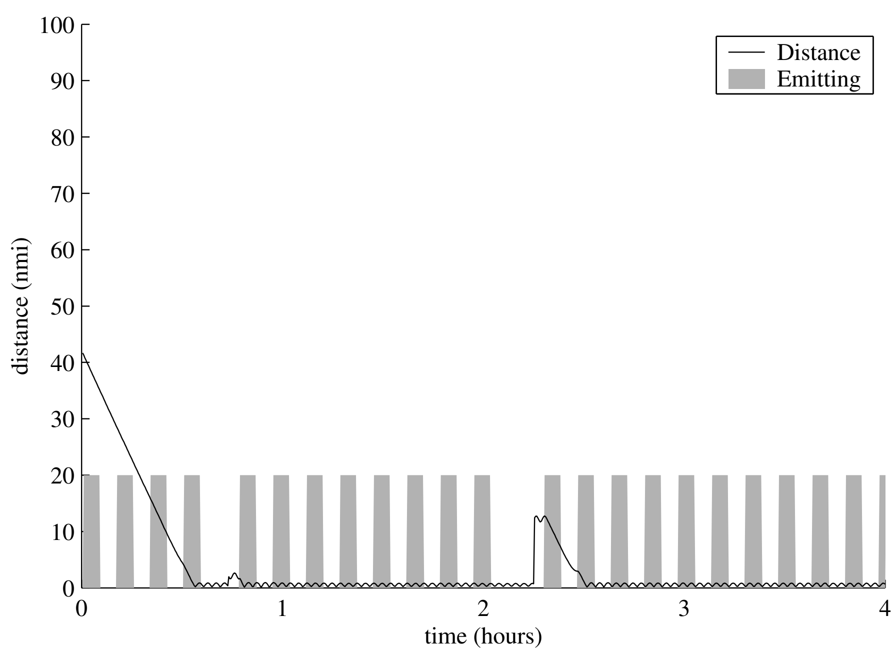
Figure 5.24: Histogram of the number of successful controllers for each evolutionary run for intermittently emitting, mobile radars.

to the second position, the UAV responds quickly. When the radar moves to the third position, it takes the UAV a little more time to begin circling because the radar is off when the UAV gets close. The flight path shown in Figure 5.26 looks very similar to the behavior shown by UAVs flying to continuously emitting, mobile radars. In this case, the periodic signal does not hinder the UAVs ability to find the target. In Figure 5.27, the UAV is heading toward the radar's first position when the radar moves. After circling the second position, the UAV moves to follow the radar to the third position. The UAV never has time to move to the radar's fourth position because the simulation ends. Similarly, Figure 5.28 shows a case where the UAV overshoots slightly while finding new radar positions. This is due to the intermittently emitting radar. Figure 5.29 shows a UAV that flies directly to the radar's initial position and once the radar moves, the UAV flies directly to the new position. All five of these flight paths show how fit the evolved controllers are in flying UAVs to intermittently emitting, mobile radars.

It should not be surprising that evolution did so poorly on this experiment. The radar type

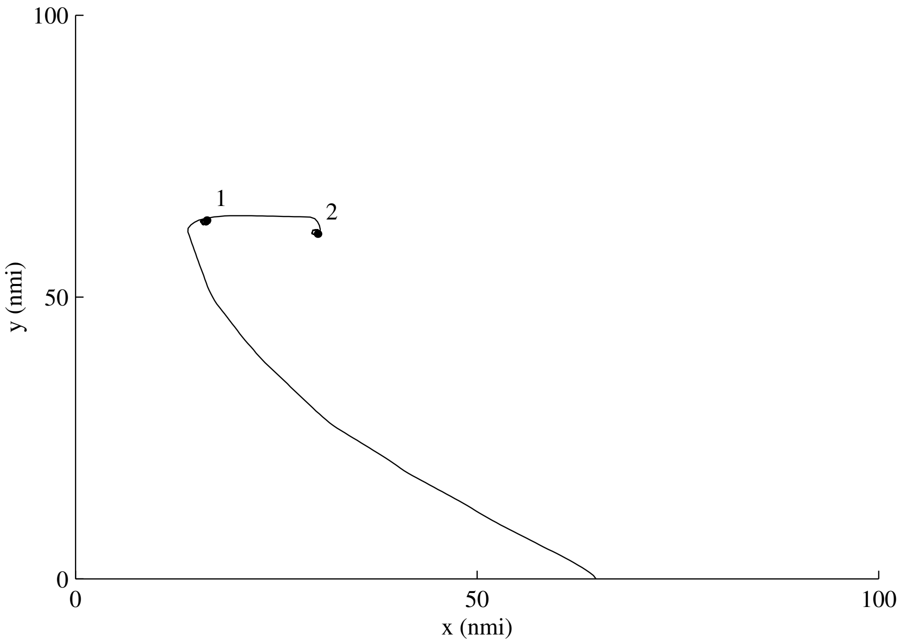


(a)

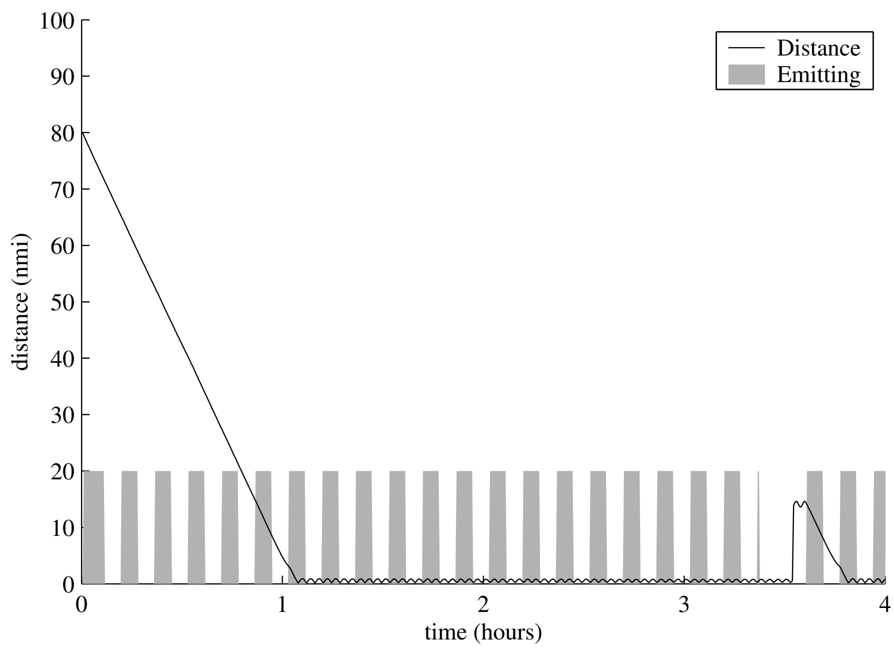


(b)

Figure 5.25: Flight path 1 for a UAV controller to an intermittently emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

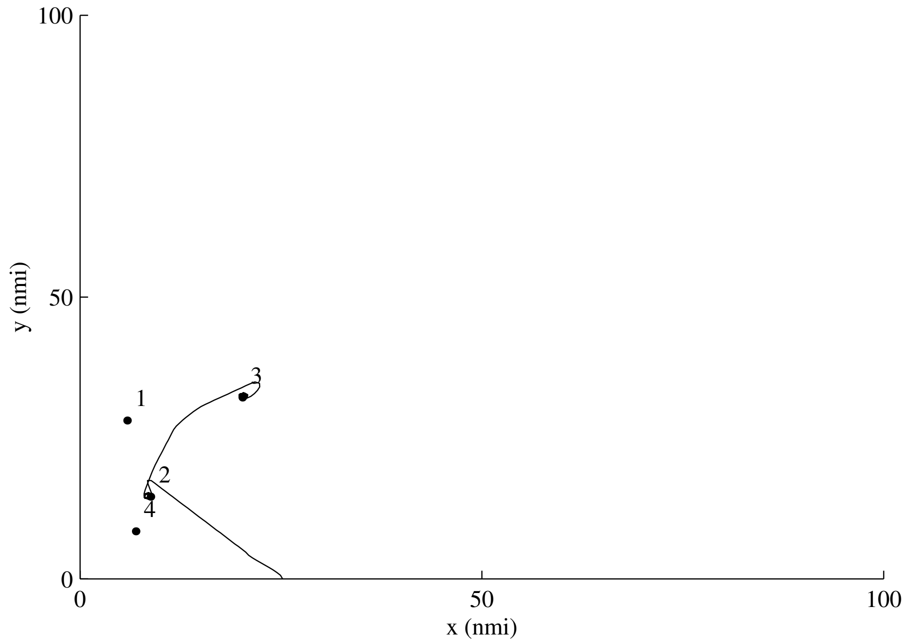


(a)

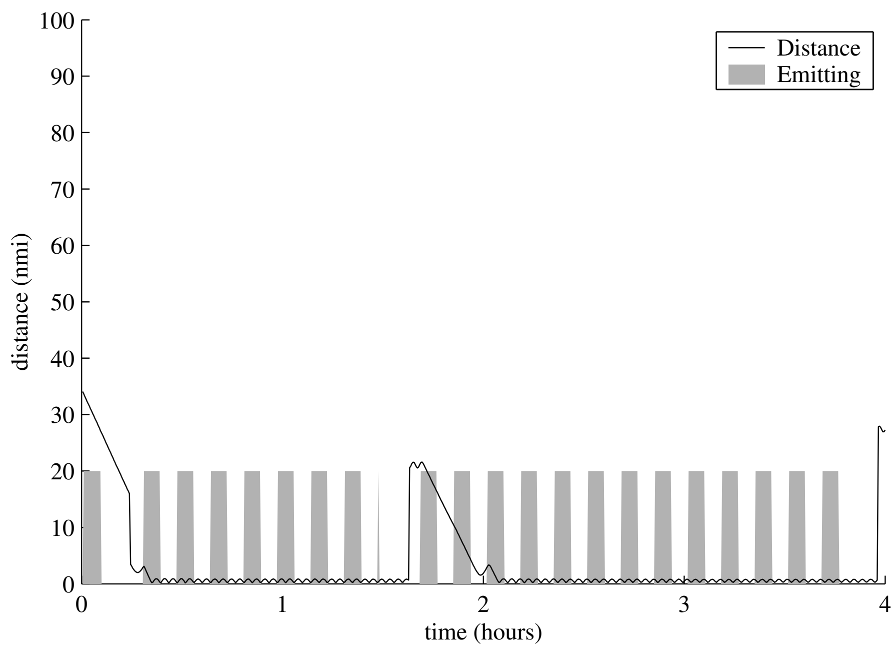


(b)

Figure 5.26: Flight path 2 for a UAV controller to an intermittently emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

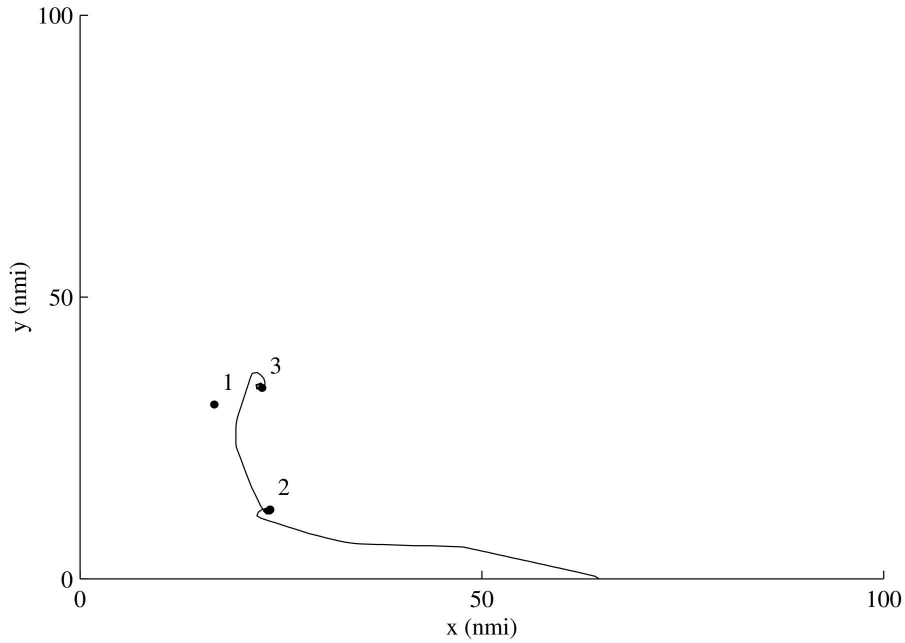


(a)

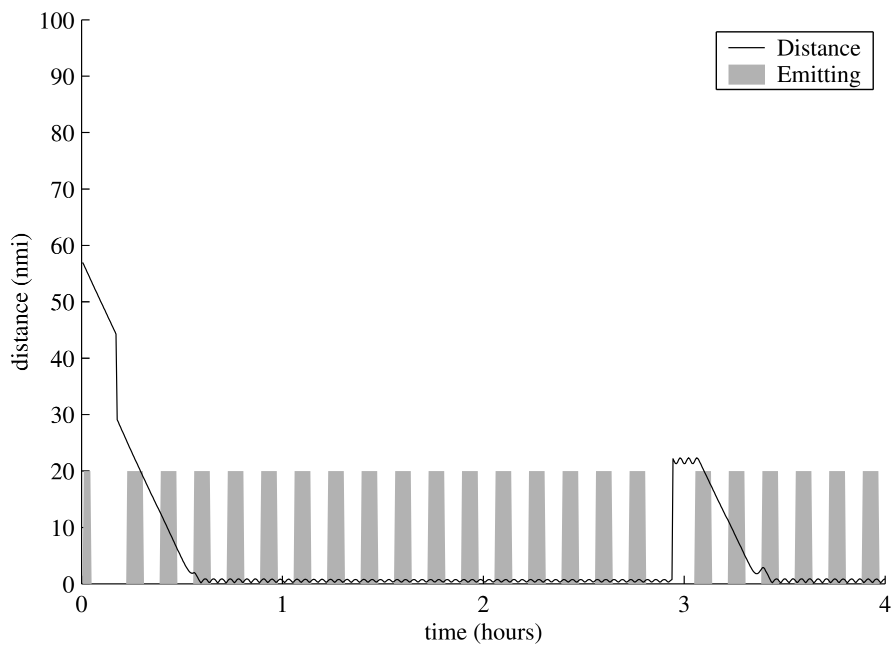


(b)

Figure 5.27: Flight path 3 for a UAV controller to an intermittently emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

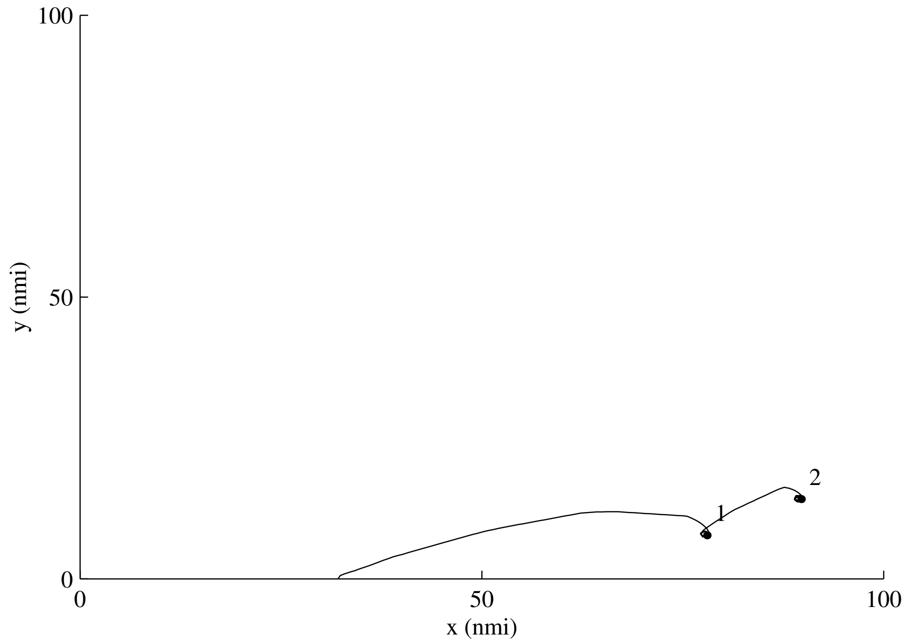


(a)

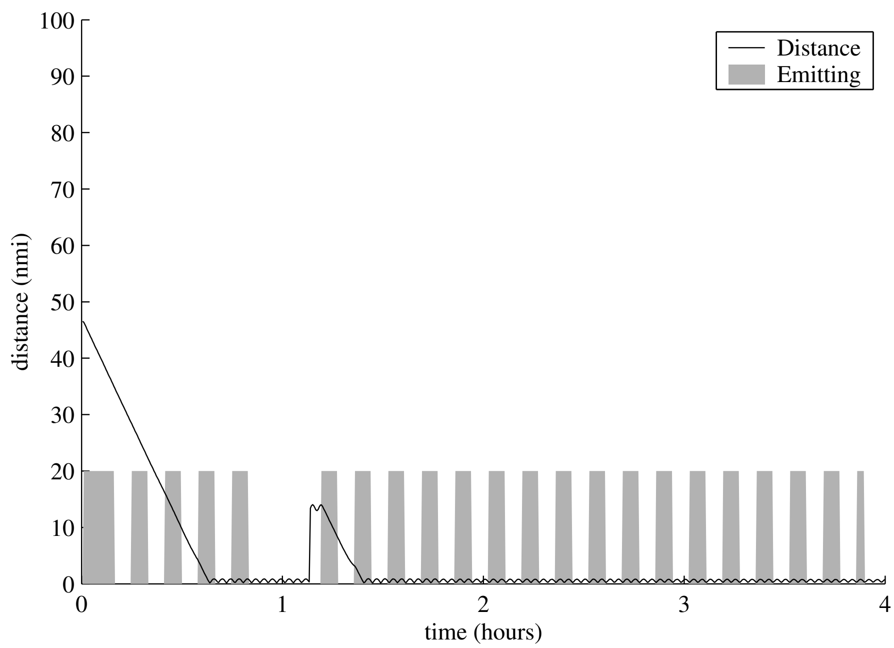


(b)

Figure 5.28: Flight path 4 for a UAV controller to an intermittently emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).



(a)



(b)

Figure 5.29: Flight path 5 for a UAV controller to an intermittently emitting, mobile radar (a), the distance between the UAV and the radar, and the emitting period and duration of the radar (b).

used, an intermittently emitting, mobile radar, combined two previous experiments that were difficult for evolution. Attempting to evolve a fit controller from a random initial population for this radar is very difficult because a UAV receives sensor information less than half the time. Evolution must be able to devise complex strategies, something it found difficult to do in this experiment.

5.4 Incremental Evolution

Environmental incremental evolution was used in an attempt to improve the chances of multi-objective genetic programming successfully evolving acceptable controllers for the more complex radar types. Unlike the direct evolution experiments, which always started with a random initial population, these experiments used evolved populations from simpler radar types as seed populations. The evolution of a seed population using continuously emitting, stationary radars is presented. Then, incremental evolution results are presented for several radar types: 1) intermittently emitting, stationary radars evolved from the seed population, 2) continuously emitting, mobile radars evolved from the seed population, 3) intermittently emitting, stationary radars evolved in multiple increments, and 4) intermittently emitting, mobile radars evolved in multiple increments. Incremental evolution greatly aided the evolution of fit controllers for complex radar types.

5.4.1 Seed Populations

The first stage of evolution for the incremental evolution experiments was evolving seed populations on the continuously emitting, stationary radar. This experiment was set up identically to the experiment described in Section 5.3.1. This new set of evolutionary runs was used not

Table 5.13: Results for seed population experiments evolved on continuously emitting, stationary radars.

Number of evolutionary runs	50
Number of successful runs	45
Success percentage	90%
Total number of successful controllers	2,815
Average successful controllers per run	56.30
Maximum successful controllers for a run	166
Minimum successful controllers for a run	1

only as a seed for incremental evolution experiments, it was also used to verify the previous experiment on continuously emitting, stationary radars.

In this experiment, 50 new evolutionary runs were performed. For each run, a new population was randomly initialized and then evolved for 600 generations. The results for this experiment are summarized in Table 5.2. Like the previous experiments using continuously emitting, stationary radars, 45 of the 50 evolutionary runs were successful, for a success rate of 90 percent. The number of acceptable controllers evolved during a successful run ranged from 1 to 166. Figure 5.30 shows a histogram of the number of acceptable controllers evolved in each successful run. Over all 50 runs, 2,815 acceptable controllers were evolved for an average of 56.30 successful controllers per evolutionary run.

5.4.2 Intermittently Emitting, Stationary Radar

In this experiment, 50 evolutionary runs were performed. Each of the seed populations evolved in Section 5.4.1 was used as the initial population for an evolutionary run. These incremental evolution runs used intermittently emitting, stationary radars with regular periods. The seeded populations were evolved for 400 generations on all four fitness functions, as described in Section 4.3.2. The evolutionary process is shown in Figure 5.31. This figure shows that in

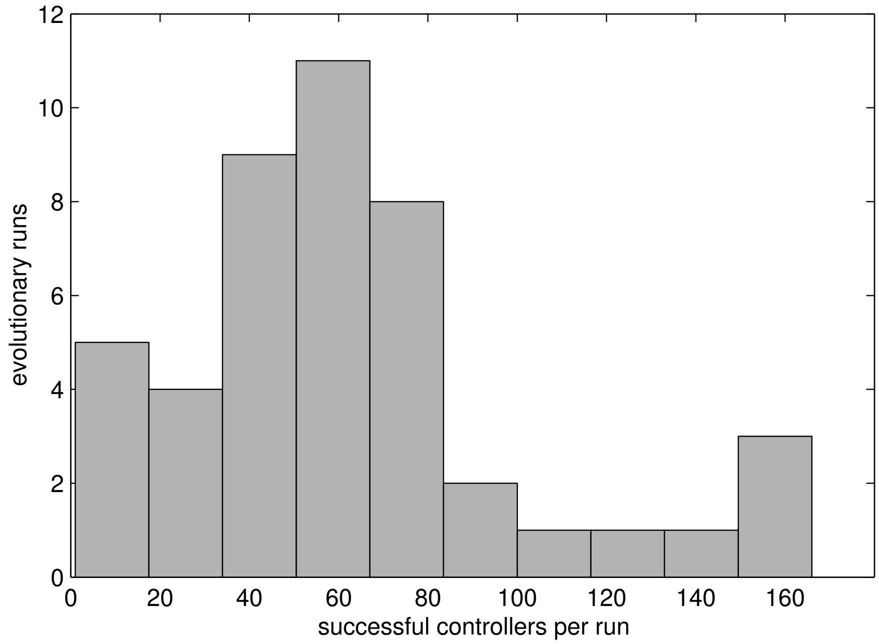


Figure 5.30: Histogram of the number of successful controllers for each evolutionary run of the seed populations using continuously emitting, stationary radars.

the first stage of evolution, the population is randomly initialized and the radar type used was continuously emitting and stationary. The evolved population was used in the second stage of evolution and the radar type was intermittently emitting and stationary. The results of these evolutionary runs are summarized in Table 5.14. The use of incremental evolution improved the success rate of evolution on this type of radar. Of the 50 evolutionary runs, 34 were successful, for a 68 percent success rate. The number of successful controllers evolved during a run ranged from 1 to 184. Figure 5.32 shows a histogram of the number of acceptable controllers evolved in each successful run. While there were still many runs that only produced a few successful controllers, many runs did appear to propagate good subtrees effectively. A total of 2,526 acceptable controllers were evolved for an average of 50.52 successful controllers per evolutionary run.

Using incremental evolution to evolve controllers for intermittently emitting, stationary radars was more successful than simply using direct evolution. The success rate of this evolutionary

Table 5.14: Results for incremental evolution experiments evolved on intermittently emitting, stationary radars.

Number of evolutionary runs	50
Number of successful runs	34
Success percentage	68%
Total number of successful controllers	2,526
Average successful controllers per run	50.52
Maximum successful controllers for a run	184
Minimum successful controllers for a run	1

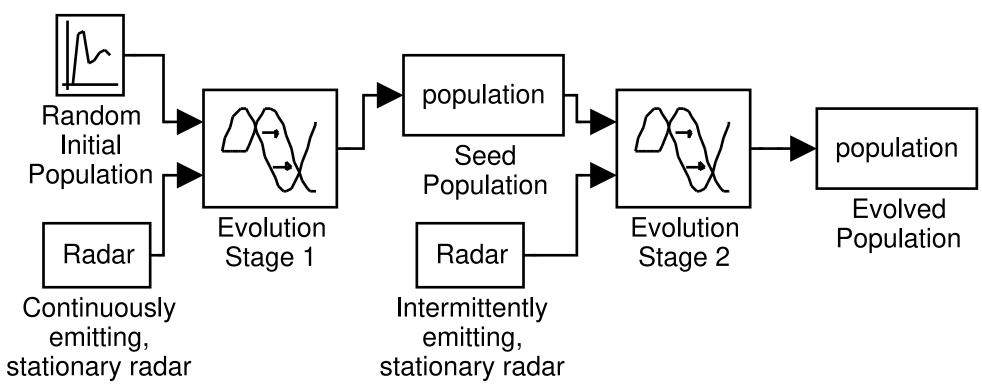


Figure 5.31: Evolutionary process for incremental evolution of controllers for intermittently emitting, stationary radars.

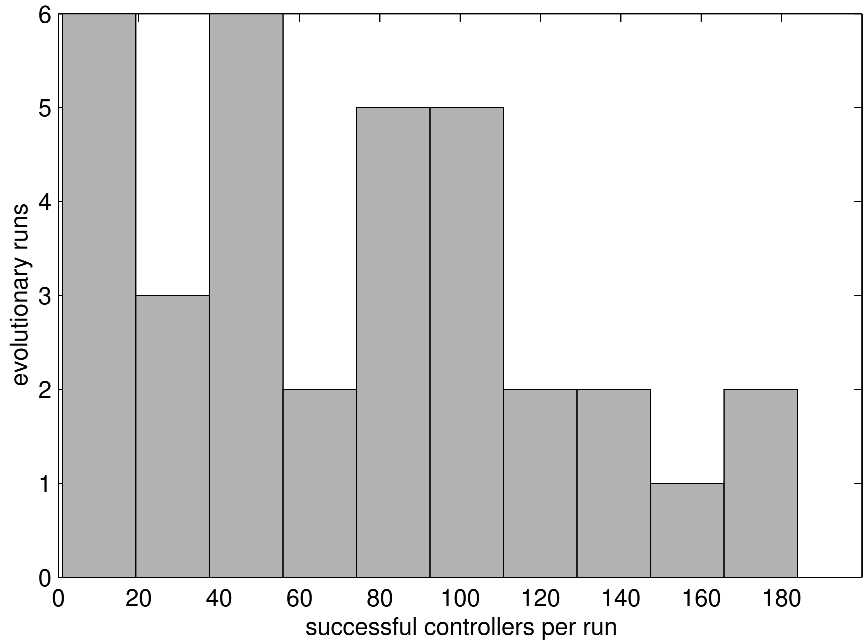


Figure 5.32: Histogram of the number of successful controllers incrementally evolved on intermittently emitting, stationary radars.

system increased from 50 percent to 68 percent as the number of successful evolutionary runs increased from 25 to 34. Clearly, incremental evolution was beneficial. However, this success rate is still low, suggesting that this radar type might benefit from multiple increments of evolution. Based on all the results from intermittently emitting radars, it seems that evolution has a very difficult time creating effective strategies for intermittent emitters, where sensor data is much less frequent than for continuous emitters.

5.4.3 Continuously Emitting, Mobile Radar

For this experiment, 50 evolutionary runs were performed. The evolutionary process was similar to that for incrementally evolving controllers on intermittently emitting, stationary radars. Each of the seed populations evolved in Section 5.4.1 was used as the initial population for an evolutionary run, which evolved for 400 generations. The evolutionary process is shown in

Table 5.15: Results for incremental evolution experiments evolved on continuously emitting, mobile radars.

Number of evolutionary runs	50
Number of successful runs	45
Success percentage	90%
Total number of successful controllers	2,774
Average successful controllers per run	55.48
Maximum successful controllers for a run	179
Minimum successful controllers for a run	1

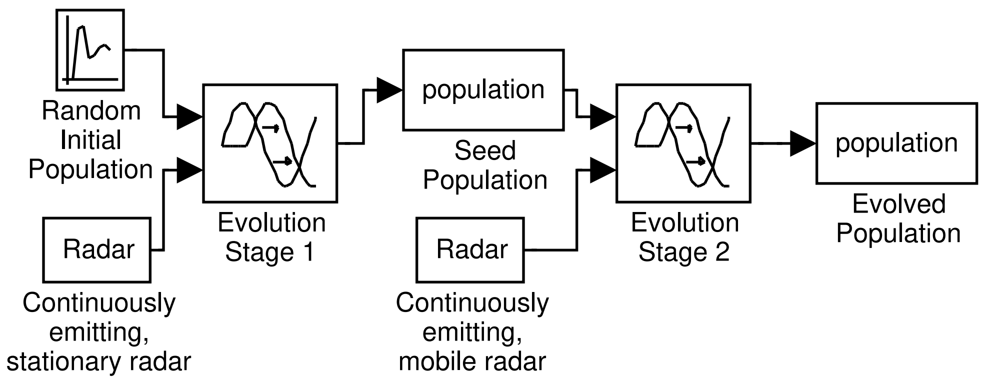


Figure 5.33: Evolutionary process for incremental evolution of controllers for continuously emitting, mobile radars.

Figure 5.33. The results of these evolutionary runs are summarized in Table 5.15. The use of incremental evolution vastly improved the success rate of evolution on this type of radar. Of the 50 evolutionary runs, 45 were successful, for a 90 percent success rate. Using incremental evolution made the success rate for continuously emitting, mobile radars equal to that of the simpler continuously emitting, stationary radars. The number of successful controllers evolved during a run ranged from 1 to 179. Figure 5.34 shows a histogram of the number of acceptable controllers evolved in each successful run. While this histogram looks very similar to the one for the direct evolution experiment on the same type of radar, in this case more of the evolutionary runs produced high numbers of acceptable controllers. A total of 2,774 acceptable controllers were evolved for an average of 55.48 successful controllers per evolutionary run.

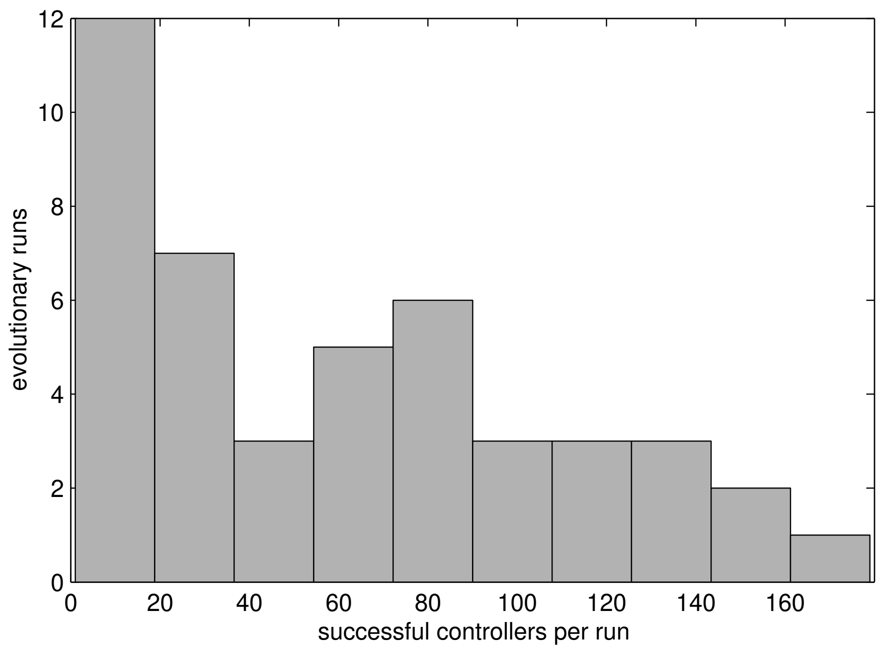


Figure 5.34: Histogram of the number of successful controllers incrementally evolved on continuously emitting, mobile radars.

Using incremental evolution to produce controllers for continuously emitting, mobile radars was extremely successful. The number of successful evolutionary runs increased from 36 to 45 for an increase in the success rate from 72 percent to 90 percent, an equal success rate to the seed populations from which these results were evolved. While the success rates for the two sets of populations were equal, the success of an evolutionary run in this experiment did not correspond directly to the success of its seed population. While it might be possible to achieve an even higher success rate for continuously emitting, mobile radars, equaling the success rate for the simplest case of continuously emitting, stationary radars is more than satisfactory.

5.4.4 Intermittently Emitting, Stationary Radar with Multiple Increments

This experiment used multiple increments of evolution to arrive at the final evolved population. Like the other experiments, 50 evolutionary runs were performed. Each of the evolved popu-

Table 5.16: Results for incremental evolution experiments evolved on intermittently emitting, stationary radars evolved in multiple increments.

Number of evolutionary runs	50
Number of successful runs	42
Success percentage	84%
Total number of successful controllers	2,083
Average successful controllers per run	41.66
Maximum successful controllers for a run	143
Minimum successful controllers for a run	1

lations from Section 5.4.3 was used as a seed population for 400 generations of evolution on intermittently emitting, stationary radars with regular periods. The full evolutionary process, as shown in Figure 5.35, started with a random initial population which was subsequently evolved on continuously emitting, stationary radars. The resulting population was used as a seed for evolving on continuously emitting, mobile radars. Then, the population was evolved on intermittently emitting, stationary radars. The results of this experiment are summarized in Table 5.16. The use of incremental evolution over multiple increments vastly improved the success rate of evolution on this type of radar. Of the 50 evolutionary runs, 42 were successful, for an 84 percent success rate. The number of successful controllers evolved during a run ranged from 1 to 143. Figure 5.36 shows a histogram of the number of acceptable controllers evolved in each successful run. A total of 2,083 acceptable controllers were evolved for an average of 41.66 successful controllers per evolutionary run.

Using multiple stages of incremental evolution vastly increased evolution's ability to successfully produce good results for this problem. The success rates for intermittently emitting, stationary radars were 50 percent for direct evolution, 68 percent for incremental evolution from the seed population, and 84 percent in this experiment. The number of successful evolutionary runs increased from 25 to 34 to 42. This vast increase in success rate suggests that incremental evolution is a very effective technique for this problem. These results also help to confirm that

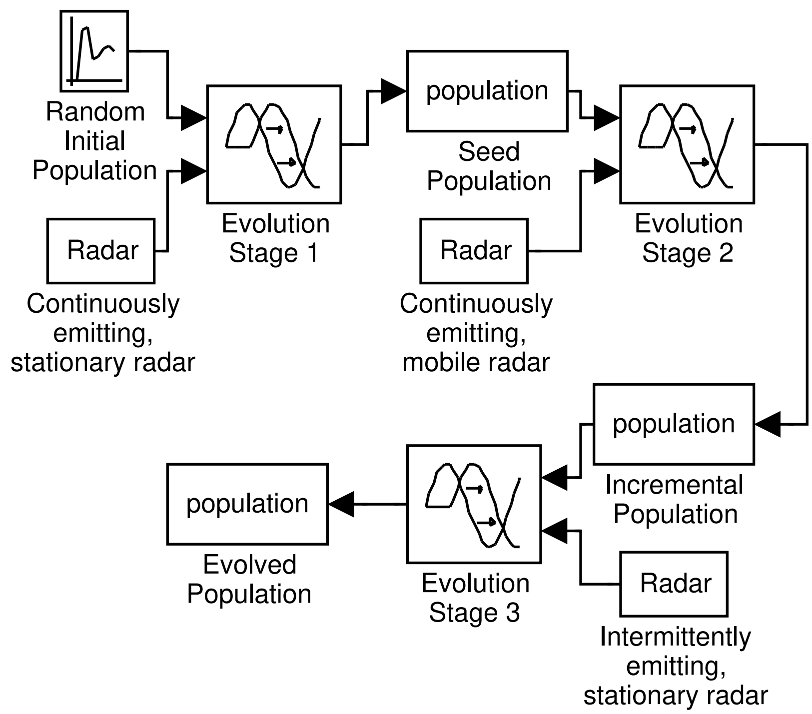


Figure 5.35: Evolutionary process for the incremental evolution of controllers for intermittently emitting, stationary radars using multiple increments.

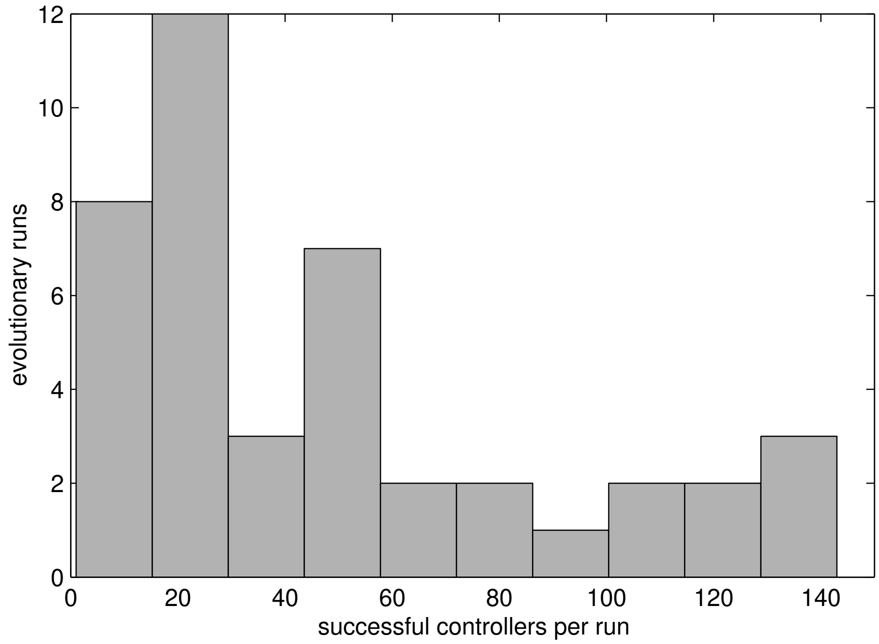


Figure 5.36: Histogram of the number of successful controllers incrementally evolved on intermittently emitting, stationary radars over multiple increments.

the lack of sensory data is the most difficult hurdle for evolution to overcome, and presenting radars in order of the percentage of time the radar is emitting is perhaps the best method for incrementally evolving controllers.

5.4.5 Intermittently Emitting, Mobile Radar with Multiple Increments

In this final experiment incrementally evolving controllers for UAVs, multiple increments of evolution were used to arrive at the final evolved population. For each evolutionary run, the seed population was taken from the final population of the experiment described in Section 5.4.4. For the 50 evolutionary runs, each of the seed populations was evolved for 400 generations on intermittently emitting, mobile radars with regular periods. The full evolutionary process, shown in Figure 5.37, started with a random initial population which was first evolved on continuously emitting, stationary radars, second on continuously emitting, mobile radars,

Table 5.17: Results for incremental evolution experiments evolved on intermittently emitting, mobile radars evolved in multiple increments.

Number of evolutionary runs	50
Number of successful runs	37
Success percentage	74%
Total number of successful controllers	1,602
Average successful controllers per run	32.04
Maximum successful controllers for a run	143
Minimum successful controllers for a run	2

and third on intermittently emitting, stationary radars. The results of this experiment are summarized in Table 5.17. The use of incremental evolution over multiple increments vastly improved the success rate of evolution on this type of radar. Of the 50 evolutionary runs, 37 were successful, for a 74 percent success rate. The number of successful controllers evolved during a run ranged from 2 to 143. Figure 5.38 shows a histogram of the number of acceptable controllers evolved in each successful run. A total of 1,602 acceptable controllers were evolved for an average of 32.04 successful controllers per evolutionary run.

The use of multiple increments, or stages of evolution, dramatically increased the ability of evolution to produce adept controllers for this type of radar. The success rate for intermittently emitting, mobile radars was 32 percent for direct evolution, but with incremental evolution, the success rate jumped all the way to 74 percent. The number of successful evolutionary runs increased from 16 to 37. This final evolutionary experiment further demonstrates the effectiveness of incremental evolution.

5.4.6 Analysis of Incrementally Evolved Controllers

Incremental evolution dramatically increased the success rates of evolution for the more complex radar types. Table 5.18 shows the results for each incremental evolution experiment.

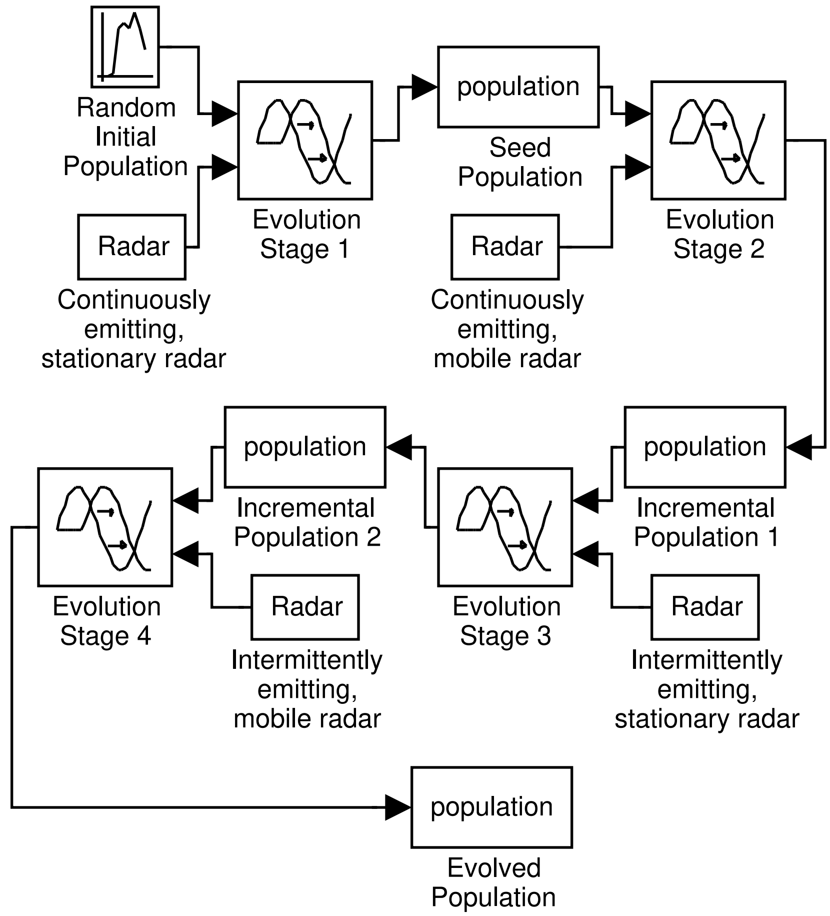


Figure 5.37: Evolutionary process for the incremental evolution of controllers for intermittently emitting, mobile radars using multiple increments.

Table 5.18: Incremental evolution experimental results

Radar Type	Runs		
	Total	Successful	Percentage
Continuous, Stationary (5.4.1)	50	45	90%
Intermittent, Stationary (5.4.2)	50	34	68%
Continuous, Mobile (5.4.3)	50	45	90%
Intermittent, Stationary (5.4.4)	50	42	84%
Intermittent, Mobile (5.4.5)	50	37	74%

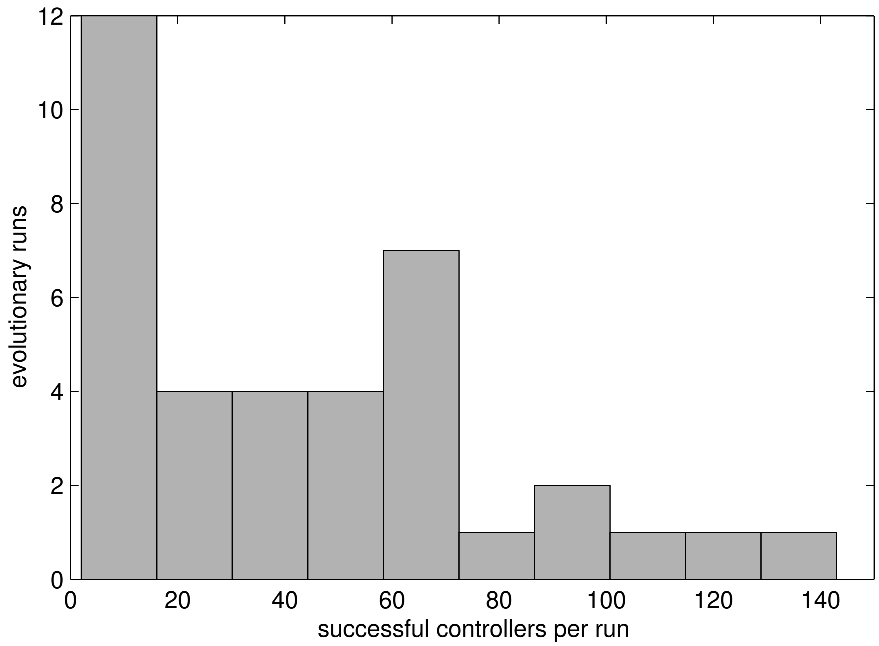


Figure 5.38: Histogram of the number of successful controllers incrementally evolved on intermittently emitting, mobile radars over multiple increments.

These results demonstrate how incremental evolution encourages the evolution of fit controller for complex problems.

A major issue in GP is code growth, or bloat. EC problems with genomes of variable length tend to grow over time, making recombination less likely to benefit the fitness of a population. Like most GP applications, the experiments presented in this research suffered from some code growth. The maximum depth of an individual in the population, for instance, quickly went to the 21, the maximum depth allowed by the GP algorithm. Early in evolution, both depth and complexity (the number of nodes in a program tree) increase dramatically, as seen in Figures 5.39 and 5.40. However, the averages of both depth and complexity tended to fluctuate in this research, rather than increasing throughout evolution. For incremental evolution in this research, where controllers were evolved for thousands of generations, code growth proved not to be an enormous problem. As mentioned in Section 4.3, diversity is often an issue when using incremental evolution. In these experiments, populations tended to remain diverse, possibly

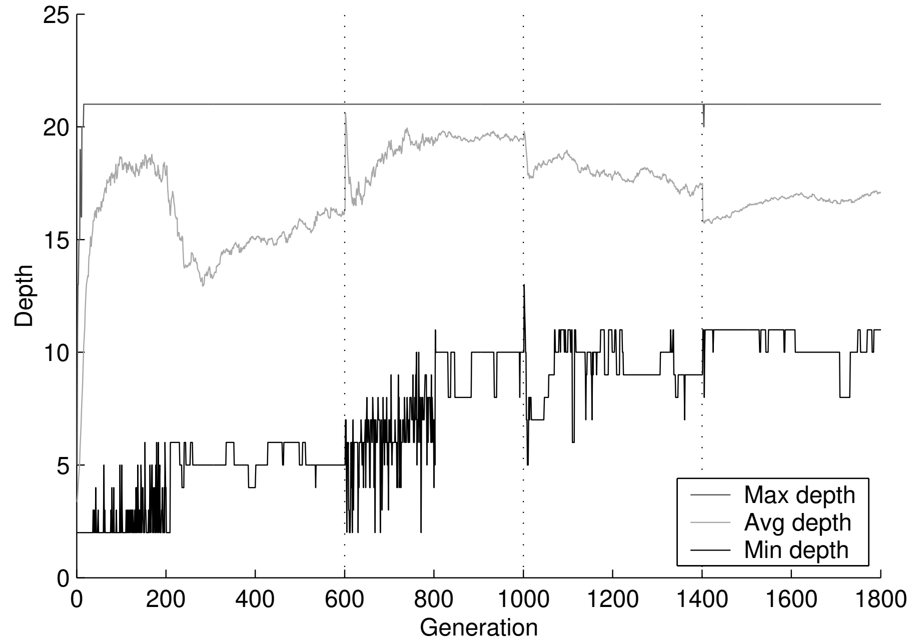


Figure 5.39: Depth by generation in the incremental evolution of controllers for intermittently emitting, mobile radars. Transitions between the stages of evolution are shown.

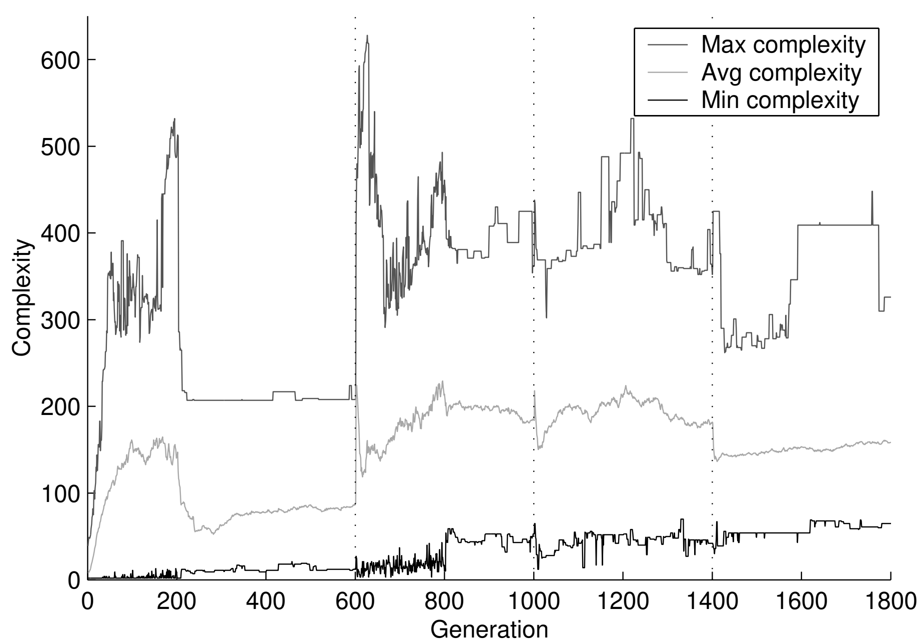


Figure 5.40: Complexity by generation in the incremental evolution of controllers for intermittently emitting, mobile radars. Transitions between the stages of evolution are shown.

because of the use of crowding distance in the multi-objective GP algorithm.

Rather than using a different controller for each radar type, it would be advantageous if the controller flying a UAV could handle multiple radar types. The controllers evolved in the experiment described in Section 5.4.5 were incrementally evolved on most of the radar types examined in this research. Of the 25,000 controllers produced over the 50 evolutionary runs of this experiment (for each of the 50 runs, 500 controllers were evolved), 1,602 were successful on intermittently emitting, mobile radars. These controllers were then evaluated on five radar types: 1) continuously emitting, stationary radars, 2) intermittently emitting, stationary radars with regular periods, 3) intermittently emitting, stationary radars with irregular periods, 4) continuously emitting, mobile radars, and 5) intermittently emitting, mobile radars. Every one of the 1,602 controllers was successful for each of the five radar types. Incremental evolution produced a set of controllers that could be used for any of the radar types in this research. When autonomous navigation controllers are used to fly real UAVs, it is essential to have a single controller that can handle multiple radar types. Based on the information available to the UAV, it is difficult to know what kind of radar the UAV is approaching, and it is far easier to have one robust controller that is used all the time rather than switching between several simpler controllers. The evolution of controllers able to handle many different tasks demonstrates another advantage to incremental evolution.

5.5 Transference to a Wheeled Mobile Robot

The main goal of this research was to develop autonomous navigation controllers that could be used to fly real UAVs. However, UAVs are very expensive, and flight tests on physical UAVs generally require the scheduling of large flight ranges. A UAV was not available for testing at the time of this research, though future tests are planned.

Rather than not attempting the transference of evolved controllers at all, controllers were tested on a wheeled mobile robot. This experiment used one of the robots in a colony of small, computationally powerful, mobile robots designed for ER research in the Center for Robotics and Intelligent Machines at North Carolina State University. These robots are called EvBots [24]. The robot used in this experiment was an EvBot II [53], one of the second generations EvBots.

The EvBot II platform, shown in Figure 5.41, is 9 inches wide by 12 inches long by 10 inches high. Each of the EvBots is autonomously controlled by a PC/104 based on-board computer running a custom Linux distribution, Infinite Atom Linux 2.0, derived from Red Hat Linux 8.0. Every EvBot has a USB video camera and wireless network card on-board. The robots have two sets of treads and move using differential steering. Far more information about the EvBots can be found in [24], [53], and [57].

In addition to the computational power of the EvBot II, the main attractive feature with regard to this research was the acoustic array sensor for the EvBot II [53]. This sensor can find an angle and an amplitude for incoming sounds. In many ways, this acoustic array can be seen as similar to the sensor used in the simulation for this research, with sound used as the signal instead of radar emissions. Since the acoustic array can provide the same sensory information as the sensor used in the simulation for this research, it was possible to transfer evolved controllers to an EvBot.

One major difference between the acoustic array and the AoA sensor used in simulation was the accuracy. While the AoA sensor modeled in simulation was accurate within $\pm 10°$, the acoustic array on the EvBots is less accurate, approximately $\pm 45°$. As might be expected, controllers evolved on lower assumed levels of error were not particularly fit when error increased this much. Figure 5.42a shows a controller evolved for a continuously emitting, stationary radar with a sensor accurate within $\pm 10°$ tested with the same sensor, and Figure 5.42b shows the

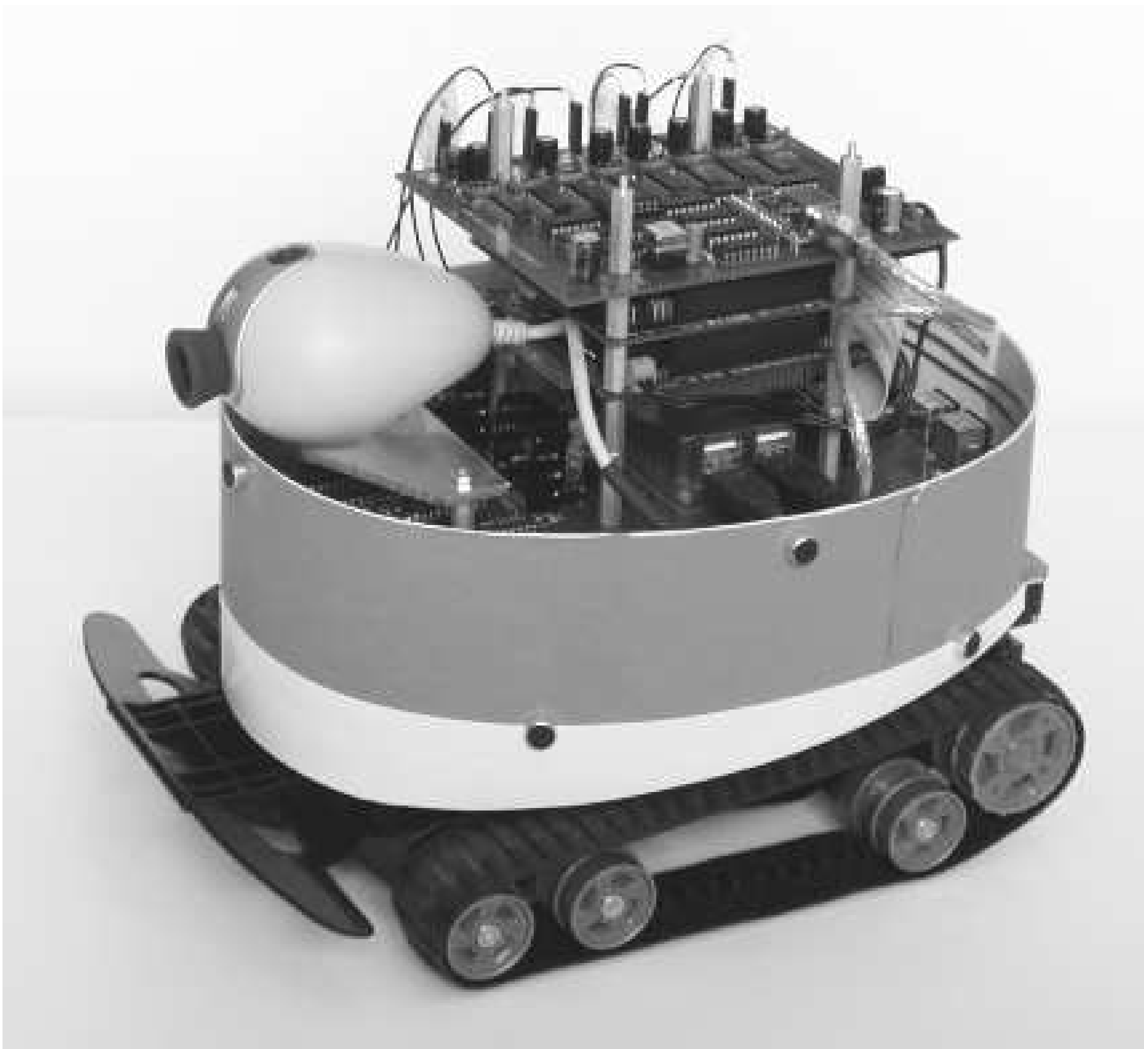
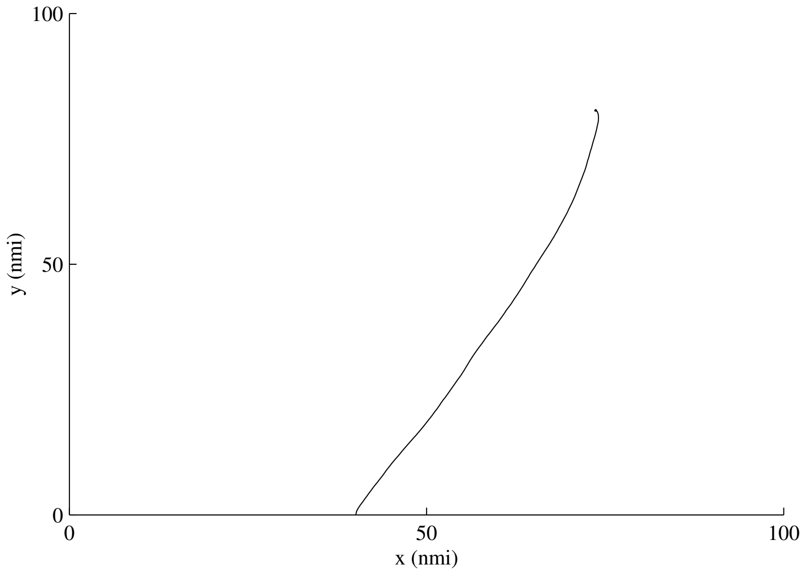


Figure 5.41: EvBot II, a small, wheeled mobile robot.

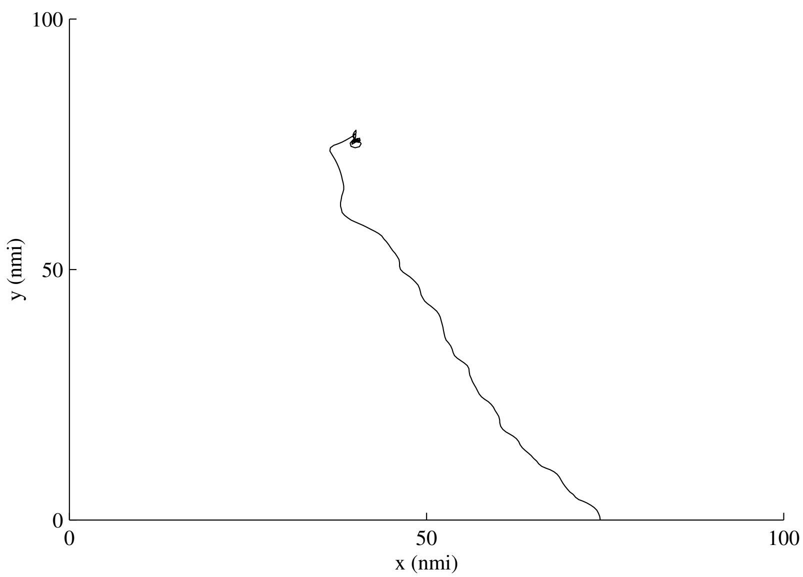
same controller tested with a sensor accurate within $\pm 45°$. Rather than using one of the controllers that had already been evolved, a new evolutionary run was done where the simulated sensor was accurate within $\pm 45°$.

The experiment was done in an arena constructed for EvBot tests. This 153 inch by 122 inch area can be set up as a maze for certain experiments, but for this experiment it was completely open. A video camera with a fisheye lens is mounted above the maze environment to document experiments. To test the controllers, a series of experiments were performed. In each experiment, the robot was placed along one wall facing toward the middle of the environment. A speaker was suspended a foot above the ground and continuously emitted a 300 Hz tone. A circle was placed directly underneath the speaker as a visual reference point, since the fisheye lens tended to distort the location of the speaker. The robot's movement was discretized into steps, much like the simulation. At each time step, the controller was executed to produce a roll angle. Since the EvBot is not controlled by a roll angle, the robot would turn to control direction. The differential drive system on the EvBot allowed all turns to happen in place, without changing the location of the robot. The EvBot was only calibrated to turn at multiples of $5°$ and the magnitude of the turn angle was always rounded down to the nearest multiple of 5. Calibrating the EvBot to turn at angles smaller than $5°$ would have been unreliable due to the size of the EvBot and the characteristics of its motors. After turning, the EvBot would always move forward the same amount, mimicking the constant speed of the UAV in simulation. The EvBot moved 3 inches per time step, and in simulation the UAV moved 0.0222 nautical miles per time step. If these values are used to scale the maze environment, then the maze would represent an area approximately 1.13 nmi by 0.90 nmi. Hence, these experiments were not testing the entire flight path, only the very end of flight when the vehicle nears the target.

Two evolved controllers were each tested 5 times on an EvBot II. The controllers were chosen based on good fitness values for normalized distance and circling distance, though level time and turn cost were also used. Both of these controllers were able to successfully drive the



(a)



(b)

Figure 5.42: Flight paths for a UAV controller to a continuously emitting, stationary radar using a sensor accurate within (a) $\pm 10°$ and (b) $\pm 45°$.

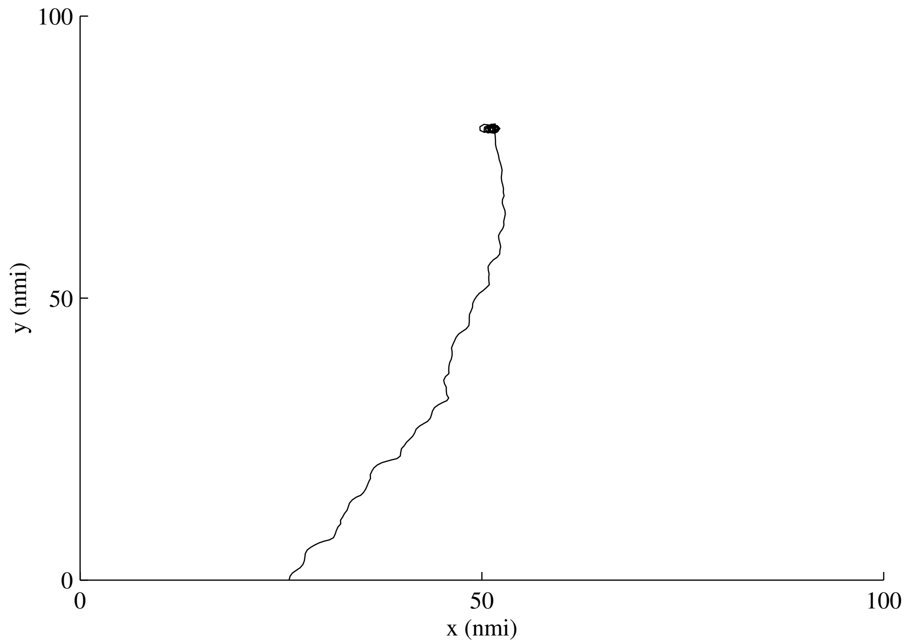


Figure 5.43: Flight path in simulation for Controller 1 to a continuously emitting, stationary radar using a sensor accurate within $\pm 45°$.

EvBot from its starting position to the speaker and then circle around the speaker. The small number of tests was enough to confirm that the controllers were consistently able to perform the task as desired.

A simulated flight path for the first controller, Controller 1, is shown in Figure 5.43. This controller had good fitness values, though it is clear from the figure that the controllers evolved with a more inaccurate sensor were not as well adapted as those shown in the rest of the research. When this controller was transferred to the EvBot, similar behavior is exhibited. Figure 5.44 shows the path from one experiment where this controller was used to navigate the robot.

Figure 5.45 shows a simulated flight path for Controller 2. This controller also had good fitness values, though the level time fitness value was lower for this controller than it was for Controller 1. This controller was very successful when transferred to the EvBot. Figure 5.46

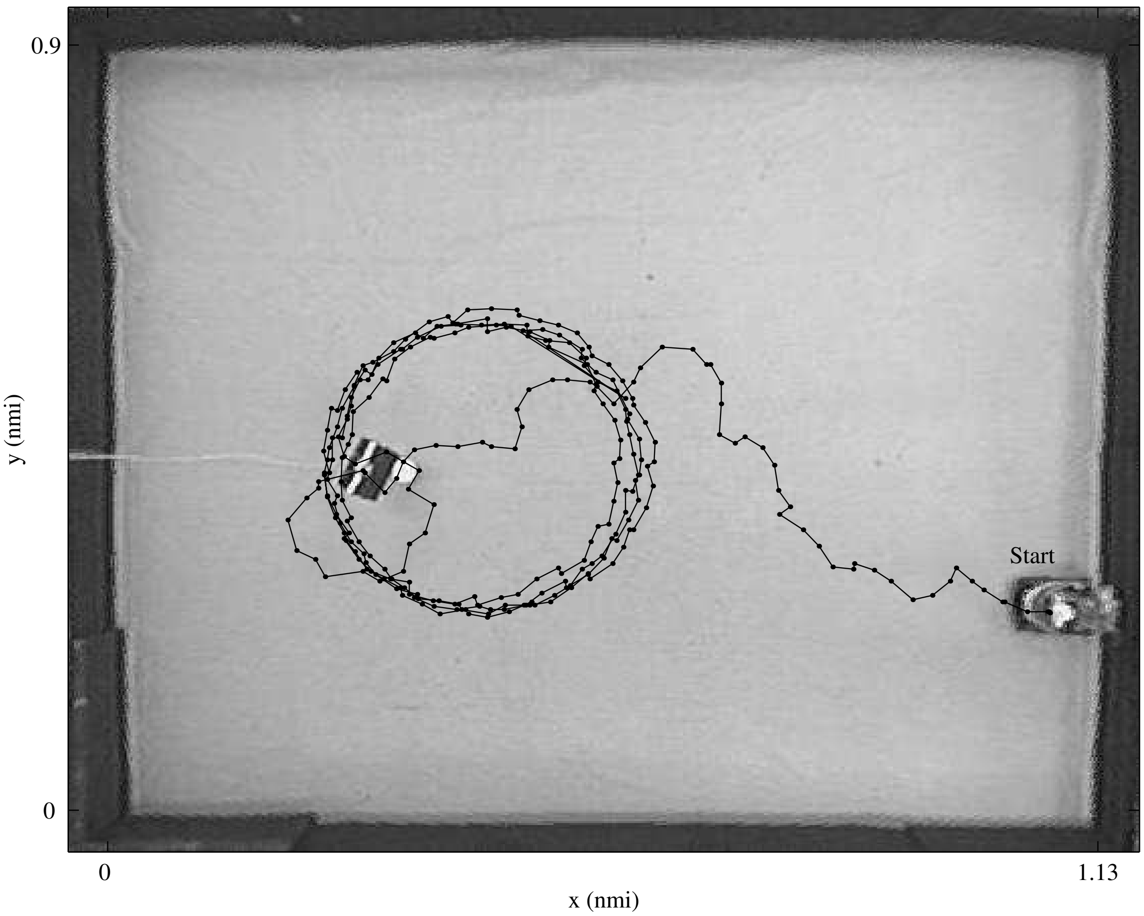


Figure 5.44: Path for an EvBot running Controller 1 moving to a continuously emitting, stationary speaker using a real acoustic array sensor.

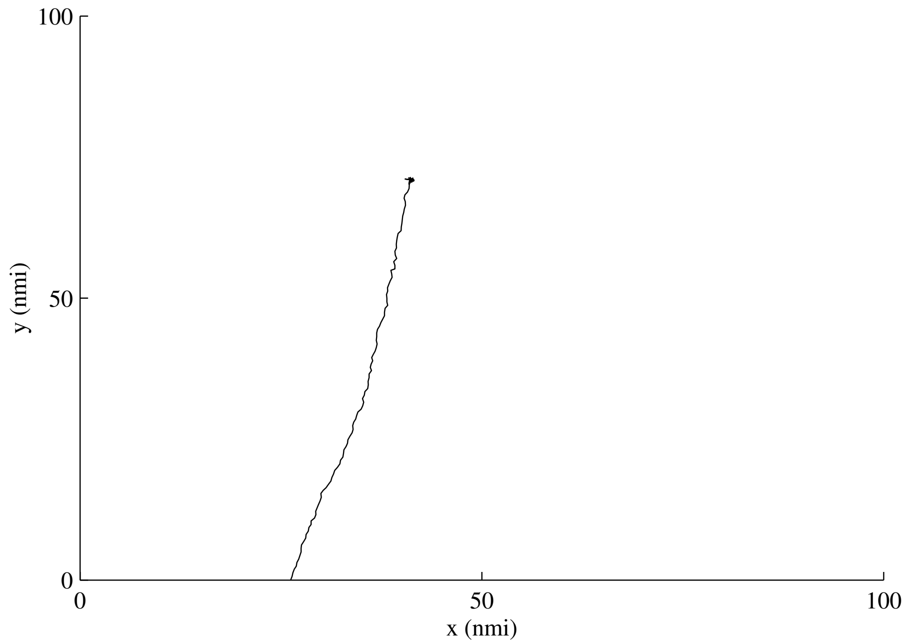


Figure 5.45: Flight path in simulation for Controller 2 to a continuously emitting, stationary radar using a sensor accurate within $\pm 45°$.

shows a path from one of the experiments using this controller. A comparison of Figure 5.46 to Figure 5.5 shows how the circling behavior of the controller on the EvBot is very similar to the controller in simulation.

These two controllers have less efficient paths than some of the other controllers evolved in this research because the sensor was less accurate. However, the paths in simulation and on the real robots are good. One sign of an extremely fit controller is a centering of the emitter when the controller begins to circle. These controllers are less well evolved because of the less accurate sensor, so the speaker is not perfectly centered inside the circling area.

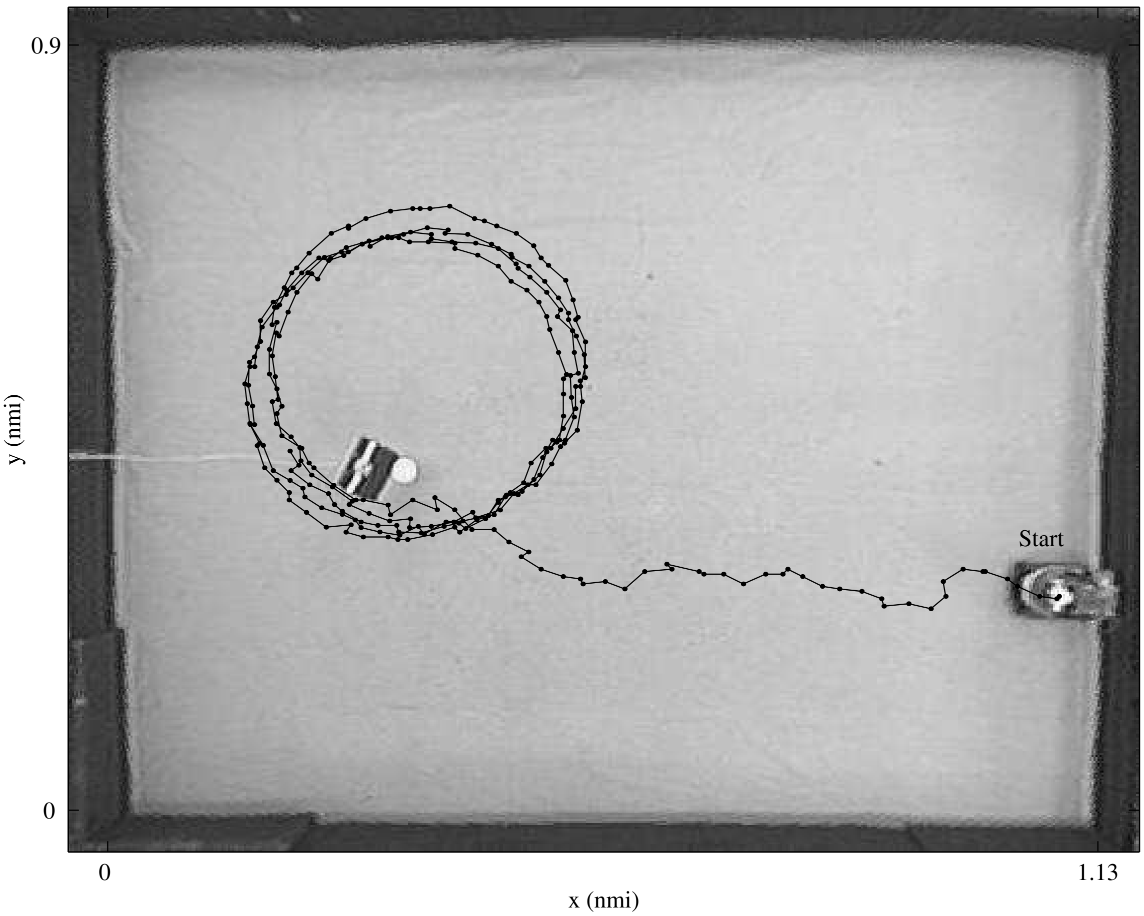


Figure 5.46: Path for an EvBot running Controller 2 moving to a continuously emitting, stationary speaker using a real acoustic array sensor.

Chapter 6

Conclusion and Future Research

6.1 Conclusions

In this research, genetic programming and multi-objective optimization were used to evolve autonomous navigation controllers for unmanned aerial vehicles. These controllers were able to fly to a target radar site, maintain an efficient flight path, and then circle around the radar. UAV controllers were evolved to be able to accomplish this task for a wide variety of radar types using inaccurate sensors in a noisy environment.

UAV navigation controllers were evolved in simulation. The control architecture used in this research was designed to be easily transferable to a large number of UAV platforms. The evolved controllers were only responsible for navigation. Low-level control was handled by an autopilot. A variety of radar types could be modeled by the simulation. Radars were described primarily by mobility and the pattern of the signal emitted by the radar. The sensors on-board the simulated UAV used in this research were able to detect the angle and amplitude of incoming electromagnetic signals. These sensors were modeled to be inaccurate, since real sensors are not ideal.

Multi-objective GP was used to evolve the navigation controllers. Parameters were selected to promote the evolution of fit controllers and to allow for parallel evaluation of individuals in the population. Both functional and environmental incremental evolution were used in this research. Four fitness functions were designed to promote the evolution of controllers able to fly to a radar, circle closely around it, and maintain an efficient flight path. The four fitness functions were 1) normalized distance, 2) circling distance, 3) level time, and 4) turn cost. Rather than focusing on optimization, the fitness functions were behavioral, intended to promote certain types of behavior in the final controllers.

The four fitness functions used for this research were sufficient to produce the desired behaviors, and all four measures were necessary to produce fit controllers. Since multi-objective optimization produces a Pareto front of solutions, rather than a single best solution, a method was needed to gauge the performance of evolution. Performance was measured by selecting a set of cutoff values for each fitness function that defined a successful controller. Two sets of experiments were successfully able to evolve navigation controllers for UAVs.

The first set of experiments started with random initial populations and evolved controllers for 5 types of radars: 1) continuously emitting, stationary radars, 2) intermittently emitting, stationary radars with regular periods, 3) intermittently emitting, stationary radars with irregular periods, 4) continuously emitting, mobile radars, and 5) intermittently emitting, mobile radars. For each of these direct evolution experiments, the GP algorithm was able to successfully evolve programs to accomplish the task. Each experiment was made up of 50 evolutionary runs, and as the radar type became more complex, the percentage of evolutionary runs that was able to produce successful controllers decreased. The continuously emitting, stationary radar was the simplest radar type for evolution, and the intermittently emitting, mobile radar was the most difficult.

The second set of experiments used environmental incremental evolution to improve the ability of the evolutionary system to produce acceptable controllers. Five experiments were performed: 1) direct evolution of a seed population on continuously emitting, stationary radars, 2) incremental evolution on intermittently emitting, stationary radars from the seed population, 3) incremental evolution on continuously emitting, mobile radars from the seed population, 4) incremental evolution on intermittently emitting, stationary radars from the population evolved on continuously emitting, mobile radars, and 5) incremental evolution on intermittently emitting, mobile radars from the last population. Again, each experiment was made up of 50 evolutionary runs. The use of incremental evolution significantly increased the success rates for the more difficult radar types. The controllers evolved for intermittently emitting, mobile radars were able to successfully accomplish the task for any radar type.

The goal of this research is to fly physical UAVs using evolved controllers. Methods were used to aid in the transference of evolved controllers to real UAVs. Simulated UAVs had inaccurate sensors and operated in a noisy environment. To test the transference of the evolved controllers, the navigation controllers evolved for UAVs were tested on a wheeled mobile robot. The results from this experiment demonstrate that evolved controllers are capable of transference to real physical vehicles.

The results of the experiments pursued in this research were very encouraging. Evolution was able to produce large numbers of autonomous navigation controllers capable of flying to a radar, circling around the radar, and maintaining an efficient flight path. Controllers were evolved for a variety of radar types, and the use of incremental evolution increased evolution's chances of evolving fit controllers.

6.2 Future Research

A major direction for future research will be the transference of evolved controllers to physical UAVs. Evolved controllers will be used in multiple flight tests. The first test will use a single UAV and a single continuously emitting, stationary radar. Additional flight tests will use more complex radar types. Flight tests may also be done with multiple UAVs and multiple radars. The transference of the controllers developed in this research to real UAVs should be relatively straightforward, as demonstrated by the transference experiments presented in Section 5.5. To use a controller evolved in this research, a UAV would need to have an autopilot that can accept roll angle as input and a sensor capable of providing the angle and amplitude of an incoming signal. The experiments presented as part of this research also suggest that these controllers are easily transferable to embedded hardware.

This research has focused primarily on situations where a single UAV acts against a single radar. In a more useful scenario, multiple UAVs would cooperate to act on multiple radars. In the near term, research will focus on evolving UAV navigation controllers for distributed multi-agent tasks. In this expanded work, the basic task for each UAV remains the same. Each UAV should find a radar and circle closely around it while maintaining an efficient flight path. However, the global task will expand from responding to a single radar to spreading the available UAVs across the target radars. The difficulty of the problem will increase as communication capabilities are added to the function and terminal sets for GP. A variety of radar types will be tested, potentially including a heterogeneous group of radars. Based on the results from the incremental evolution experiments described in this work, future work will use incremental evolution to attempt to overcome these added difficulties.

References

[1] Christoph Adami. *Introduction to Artificial Life*. Springer-Verlag, New York, 1998.

[2] David Adamy. *EW 101: A First Course in Electronic Warfare*. Artech House, 2001.

[3] Ronald C. Arkin. *Behavior-based Robotics*. MIT Press, 1998.

[4] Hezi Avraham, Gal Chechik, and Eytan Ruppin. Are there representations in embodied evolved agents? taking measures. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors, *Advances in Artificial Life - Proceedings of the 7th European Conference on Artificial Life*, 2003.

[5] Thomas Back, Ulrich Hammel, and Hans-Paul Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1), April 1997.

[6] W. Banzhaf and W. B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1):81–91, 2002.

[7] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[8] Rodney A. Brooks. Elephants don't play chess. *Robotics and Automation Systems*, 6:3–15, 1990.

[9] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[10] Rodney A. Brooks. Artificial life and real robots. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 3–10, Cambridge, MA, 1992. MIT Press.

[11] Andy Clark and Chris Thornton. Trading spaces: Computation, representation and the limits of uninformed learning. *Behavioral and Brain Sciences*, 20:57–90, 1997.

[12] Dave Cliff, Inman Harvey, and Philip Husbands. Explorations in evolutionary robotics. *Adaptive Behavior*, 2:73–110, 1993.

[13] Carlos A. Coello Coello. An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends. In *Proceedings of the Congress on Evolutionary Computation*, pages 3–13, 1999.

[14] Timothy Coffey and John A. Montgomery. The emergence of mini UAVs for military applications. *Defense Horizons*, (22):1–8, December 2002.

[15] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[16] Marc Ebner. Evolution of a control architecture for a mobile robot. In *Proceedings of the Second International Conference on Evolvable Systems*, pages 303–310, 1998.

[17] Jeffrey L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.

[18] Roger I. Eriksson. An initial analysis of the ability of learning to maintain diversity during incremental evolution. In A. A. Freitas, editor, *Data Mining with Evolutionary Algorithms*, pages 120–124, 2000.

[19] D. Filliat, J. Kodjabachian, and J.-A. Meyer. Incremental evolution of neural controllers for navigation in a 6-legged robot. In Sugisaka and Tanaka, editors, *Proceedings of the Fourth International Symposium on Artificial Life and Robots*, 1999.

[20] Gary W. Flake. *The Computational Beauty of Nature*. MIT Press, 1998.

[21] Dario Floreano and Francesco Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics*, 26(3):396–407, 1996.

[22] Dario Floreano and Stephano Nolfi. God save the red queen! competition in co-evolutionary robotics. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 398–406, 1997.

[23] Emilio Frazzoli. Maneuver-based motion planning and coordination for multiple UAVs. In *Proceedings of the AIAA/IEEE Digital Avionics Systems Conference*, 2002.

[24] John M. Galeotti. The EvBot: A small autonomous mobile robot for the study of evolutionary algorithms in distributed robotics. Master's thesis, North Carolina State University, Raleigh, NC, 2002.

[25] Geneva Aerospace. *Dakota Unmanned Aerial Vehicle*.

[26] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[27] Faustino J. Gomez and Risto Miikkulainen. Active guidance for a finless rocket using neuroevolution. In E. Cantu-Paz, editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 2084–2095, Chicago, July 2003.

[28] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: Artificial evolution, real vision. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 704–720. MIT Press, 1994.

[29] Inman Harvey, Philip Husbands, Dave Cliff, Adrian Thompson, and Nick Jakobi. Evolutionary robotics: the Sussex approach. *Robotics and Autonomous Systems*, 20:205–224, 1997.

[30] Frank Hoffmann, Tak John Koo, and Omid Shakernia. Evolutionary design of a helicopter autopilot. *3rd On-line World Conference on Soft Computing*, 1998.

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

[32] William H. Hsu and Steven M. Gustafson. Genetic programming and multi-agent layered learning by reinforcements. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, New York, July 2002.

[33] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *Proceedings of the 3rd European Conference on Artificial Life*, pages 704–720, 1995.

[34] Shotaro Kamio, Hideyuki Misuhashi, and Hitoshi Iba. Integration of genetic programming and reinforcement learning for real robots. In E. Cantu-Paz et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 470–482, Chicago, July 2003.

[35] Didier Keymeulen, Masaya Iwata, Kenji Konaka, Ryouhei Suzuki, Yasuo Kuniyoshi, and Tetsuya Higuchi. Off-life model-free and on-line model-based evolution for tracking navigation using evolvable hardware. In *Proceedings of the First European Workshop on Evolutionary Robotics*, Paris, April 1998.

[36] John Koza. *Genetic Programming*. MIT Press, 1992.

[37] John Koza. *Genetic Programming II*. MIT Press, 1994.

[38] John Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. *Genetic Programming III*. Morgan Kaufmann, 1999.

[39] John R. Koza. A hierarchical approach to learning the boolean multiplexer function. In G. Rawlins, editor, *Proceedings of Workshop on the Foundations of Genetic Algorithms and Classifier Systems*, Bloomington, IN, July 1990. Morgan Kaufmann.

[40] John R. Koza. Evolution of subsumption using genetic programming. In *Proceedings of the First European Conference on Artificial Life*, pages 110–119. MIT Press, 1992.

[41] Rajeev Kumar and Peter Rockett. Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution: A pareto converging genetic algorithm. *Evolutionary Computation*, 10(3):283–314, 2002.

[42] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multi0bjective optimization. *Evolutionary Computation*, 10(3):263–282, 2002.

[43] Wei-Po Lee, John Hallam, and Henrik Hautop Lund. Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 495–499, 1997.

[44] Hongwei Liu and Hitoshi Iba. Multi-agent learning of heterogeneous robots by evolutionary subsumption. In E. Cantu-Paz et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 1715–1728, Chicago, July 2003.

[45] Alex Lubberts and Risto Miikkulainen. Co-evolving a go-playing neural network. In *Co-evolution: Turning Adaptive Algorithms upon Themselves, Birds-of-a-Feather Workshop, GECCO*, 2001.

[46] Sean Luke and Liviu Panait. Lexicographic parsimony pressure. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 829–836, 2002.

[47] Henrik Hautop Lund and John Hallam. Evolving sufficient robot controllers. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 495–499, 1997.

[48] Joseph N. Mait and Jon G. Grossman. Relevancy and risk: The U.S. Army and future combat systems. *Defense Horizons*, (13):1–8, May 2002.

[49] John A. Marin, Robert Radtke, David Innis, Donald R. Barr, and Alan C. Schultz. Using a genetic algorithm to develop rules to guide unmanned aerial vehicles. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 1055–1060, Tokyo, Japan, 1999.

[50] Davide Marocco and Dario Floreano. Active vision and feature selection in evolutionary behavioral systems. In *From Animals to Animats 7*. MIT Press, 2002.

[51] Maja Mataric and Dave Cliff. Challenges in evolving controllers for physical robots. *Robotics and Automation Systems*, 19(1):67–83, November 1996.

[52] Maja J. Mataric. A distributed model for mobile robot environmental-learning and navigation. Technical Report AI-TR-1228, MIT Artificial Intelligence Laboratory, May 1990.

[53] Leonardo Serra Mattos. The EvBot II. Master's thesis, North Carolina State University, Raleigh, NC, 2003.

[54] Jean-Arcady Meyer, Stephane Doncieux, David Filliat, and Agnes Guillot. Evolutionary approaches to neural control of rolling, walking, swimming and flying animats or robots. In Richard J. Duro, Jose Santos, and Manuel Grana, editors, *Biologically Inspired Robot Behavior Engineering*, volume 109 of *Studies in Fuzziness and Soft Computing*, chapter 1, pages 1–43. Physica-Verlag, 2003.

[55] Robin R. Murphy. *Introduction to AI Robotics*. MIT Press, 2000.

[56] Hiroshi Nakamura, Akio Ishiguro, and Yoshiki Uchikawa. Evolutionary construction of behavior arbitration mechanisms based on dynamically-rearranging neural networks. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 158–165, 2000.

[57] Andrew L. Nelson. *Competitive Relative Performance and Fitness Selection for Evolutionary Robotics*. PhD thesis, North Carolina State University, Raleigh, NC, 2003.

[58] Andrew L. Nelson, Edward Grant, Gregory Barlow, and Mark White. Evolution of complex autonomous robot behaviors using competitive fitness. In *Proceedings of the IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, Boston, MA, September 2003.

[59] Andrew L. Nelson, Edward Grant, Gregory J. Barlow, and Thomas C. Henderson. A colony of robots using vision sensing and evolved neural controllers. In *Proceedings of the IEEE Conference on Intelligent Robots and Systems*, Las Vegas, October 2003.

[60] Andrew L. Nelson, Edward Grant, and Thomas C. Henderson. Competitive relative performance evaluation of neural controllers for competitive game playing with teams of real mobile robots. In *Proceedings of the 2002 PerMIS Workshop*, pages 43–50, August 2002.

[61] Nils J. Nilsson. Shakey the robot. Technical Note 323, SRI AI Center, Menlo Park, CA, 1984.

[62] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics*. MIT Press, 2000.

[63] Stephano Nolfi and Dario Floreano. Co-evolving predator and prey robots: Do "arms races" arise in artificial evolution? *Artificial Life*, 4(4):311–335, 1998.

[64] Stephano Nolfi, Dario Floreano, Orazio Miglino, and Francesco Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In Rodney A. Brooks and Pattie Maes, editors, *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Cambridge, MA, July 1994. MIT Press.

[65] Choong K. Oh and Gregory J. Barlow. Autonomous controller design for unmanned aerial vehicles using multi-objective genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, Portland, OR, June 2004.

[66] Peter Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, Inc., 1996.

[67] Liviu Panait and Sean Luke. Methods for evolving robust programs. In E. Cantu-Paz et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 1715–1728, Chicago, July 2003.

[68] H. Van Dyke Parunak, Michael Purcell, and Robert O'Connell. Digital pheromones for autonomous coordination of swarming uav's. In *Proceedings of the First AIAA Unmanned Aerospace Vehicles, Systems, Technologies, and Operations Conference*, April 2002.

[69] Katya Rodriguez-Vazquez, Carlos M. Fonseca, and Peter J. Fleming. Multiobjective genetic programming: A nonlinear system identification application. In *Late Breaking Papers at the 1997 Genetic Programming Conference*, pages 207–212, 1997.

[70] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[71] D. Curtis Schleher. *Introduction to Electronic Warfare*. Artech House, 1986.

[72] H. Shim, Tak John Koo, Frank Hoffmann, and S. Sastry. A comprehensive study of control design for an autonomous helicopter. In *Proceedings of the IEEE Conference on Decision and Control*, December 1998.

[73] Sara Silva and Jonas Almeida. Dynamic maximum tree depth: A simple technique for avoiding bloat in tree-based gp. In E. Cantu-Paz et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2003)*, pages 1776–1787, Chicago, July 2003.

[74] Merrill I. Skolnik. *Introduction to Radar Systems*. McGraw HIll, New York, 3rd ed. edition, 2001.

[75] Terence Soule and Robert B. Heckendorn. An analysis of the causes of code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 3(3):283–309, 2002.

[76] United States Air Force. *Aircraft Fact Sheets*.

[77] W. Grey Walter. *The Living Brain*. W. W. Norton, New York, 1953.

[78] Jay F. Winkeler and B. S. Manjunath. Incremental evolution in genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 403–411, 1998.

[79] Annie S. Wu, Alan C. Schultz, and Arvin Agah. Evolving control for distributed micro air vehicles. In *IEEE Conference on Computational Intelligence in Robotics and Automation*, November 1999.

[80] Thomas R. Yechout, Steven L. Morris, David E. Bossert, and Wayne F. Hallgren. *Introduction to aircraft flight mechanics*. AIAA Press, 2003.

Appendices

Appendix A

Experimental Results

This appendix contains results from all of the experiments presented. For each experiment, 50 evolutionary runs were completed. In this appendix, the number of successful controllers for each evolutionary run is shown, as are the summaries of evolution's performance on each experiment. The results shown here correspond to those experiments presented in Chapter 5.

A.1 Direct Evolution

A.1.1 Continuously Emitting, Stationary Radar

```
case-2003-10-10-1313-01     10
case-2003-10-10-1313-02     80
case-2003-10-10-1313-03     11
case-2003-10-10-1313-04    104
case-2003-10-10-1313-05     68
case-2003-10-10-1313-06    137
case-2003-10-10-1313-07      2
case-2003-10-10-1313-08     19
case-2003-10-10-1313-09    108
case-2003-10-10-1313-10    106
case-2003-10-10-1313-11     64
case-2003-10-10-1313-12      8
```

```
case-2003-10-10-1313-13      0
case-2003-10-10-1313-14     71
case-2003-10-10-1313-15      0
case-2003-10-10-1313-16     96
case-2003-10-10-1313-17     58
case-2003-10-10-1313-18    104
case-2003-10-10-1313-19     81
case-2003-10-10-1313-20     41
case-2003-10-10-1313-21     39
case-2003-10-10-1313-22     66
case-2003-10-10-1313-23     86
case-2003-10-10-1313-24      0
case-2003-10-10-1313-25    107
case-2003-10-10-1313-26     91
case-2003-10-10-1313-27     46
case-2003-10-10-1313-28     40
case-2003-10-10-1313-29    118
case-2003-10-10-1313-30    136
case-2003-10-10-1313-31    112
case-2003-10-10-1313-32     73
case-2003-10-10-1313-33      0
case-2003-10-10-1313-34     36
case-2003-10-10-1313-35    163
case-2003-10-10-1313-36      0
case-2003-10-10-1313-37     96
case-2003-10-10-1313-38      1
case-2003-10-10-1313-39     56
case-2003-10-10-1313-40     12
case-2003-10-10-1313-41    170
case-2003-10-10-1313-42     64
case-2003-10-10-1313-43     60
case-2003-10-10-1313-44      5
case-2003-10-10-1313-45    143
case-2003-10-10-1313-46     14
case-2003-10-10-1313-47     34
case-2003-10-10-1313-48    125
case-2003-10-10-1313-49     72
case-2003-10-10-1313-50     16

5 failures, 45 successes (success rate: 90.00%)
3149 successful controllers (mean: 62.98)
```

A.1.2 Intermittently Emitting, Stationary Radar with Regular Period

```
case-2003-10-28-1640-01     42
case-2003-10-28-1640-02    151
case-2003-10-28-1640-03     20
case-2003-10-28-1640-04      0
case-2003-10-28-1640-05      0
case-2003-10-28-1640-06      0
case-2003-10-28-1640-07     70
case-2003-10-28-1640-08     49
case-2003-10-28-1640-09      0
case-2003-10-28-1640-10     26
case-2003-10-28-1640-11      0
case-2003-10-28-1640-12      0
case-2003-10-28-1640-13     97
case-2003-10-28-1640-14      0
```

```
case-2003-10-28-1640-15     90
case-2003-10-28-1640-16    156
case-2003-10-28-1640-17     32
case-2003-10-28-1640-18      0
case-2003-10-28-1640-19      0
case-2003-10-28-1640-20     34
case-2003-10-28-1640-21      0
case-2003-10-28-1640-22      0
case-2003-10-28-1640-23      0
case-2003-10-28-1640-24      0
case-2003-10-28-1640-25     79
case-2003-10-28-1640-26    155
case-2003-10-28-1640-27     49
case-2003-10-28-1640-28     70
case-2003-10-28-1640-29      0
case-2003-10-28-1640-30      0
case-2003-10-28-1640-31     19
case-2003-10-28-1640-32      8
case-2003-10-28-1640-33     20
case-2003-10-28-1640-34      2
case-2003-10-28-1640-35      0
case-2003-10-28-1640-36      0
case-2003-10-28-1640-37      0
case-2003-10-28-1640-38      0
case-2003-10-28-1640-39    143
case-2003-10-28-1640-40    124
case-2003-10-28-1640-41      0
case-2003-10-28-1640-42    145
case-2003-10-28-1640-43      0
case-2003-10-28-1640-44    136
case-2003-10-28-1640-45     45
case-2003-10-28-1640-46      0
case-2003-10-28-1640-47      0
case-2003-10-28-1640-48      0
case-2003-10-28-1640-49    129
case-2003-10-28-1640-50      0

25 failures, 25 successes (success rate: 50.00%)
1891 successful controllers (mean: 37.82)
```

A.1.3 Intermittently Emitting, Stationary Radar with Irregular Period

```
case-2003-11-26-1111-01      0
case-2003-11-26-1111-02     79
case-2003-11-26-1111-03     91
case-2003-11-26-1111-04      0
case-2003-11-26-1111-05     10
case-2003-11-26-1111-06    108
case-2003-11-26-1111-07      0
case-2003-11-26-1111-08    125
case-2003-11-26-1111-09    102
case-2003-11-26-1111-10      0
case-2003-11-26-1111-11     72
case-2003-11-26-1111-12     98
case-2003-11-26-1111-13      0
case-2003-11-26-1111-14     70
case-2003-11-26-1111-15      0
case-2003-11-26-1111-16    109
```

```
case-2003-11-26-1111-17      0
case-2003-11-26-1111-18      0
case-2003-11-26-1111-19      0
case-2003-11-26-1111-20    110
case-2003-11-26-1111-21    141
case-2003-11-26-1111-22     99
case-2003-11-26-1111-23     62
case-2003-11-26-1111-24      0
case-2003-11-26-1111-25      0
case-2003-11-26-1111-26    102
case-2003-11-26-1111-27    134
case-2003-11-26-1111-28      0
case-2003-11-26-1111-29     68
case-2003-11-26-1111-30      0
case-2003-11-26-1111-31     38
case-2003-11-26-1111-32     41
case-2003-11-26-1111-33      0
case-2003-11-26-1111-34      0
case-2003-11-26-1111-35     18
case-2003-11-26-1111-36      0
case-2003-11-26-1111-37     93
case-2003-11-26-1111-38     31
case-2003-11-26-1111-39    172
case-2003-11-26-1111-40     99
case-2003-11-26-1111-41      0
case-2003-11-26-1111-42     22
case-2003-11-26-1111-43     76
case-2003-11-26-1111-44      0
case-2003-11-26-1111-45      0
case-2003-11-26-1111-46      0
case-2003-11-26-1111-47    104
case-2003-11-26-1111-48      8
case-2003-11-26-1111-49     92
case-2003-11-26-1111-50      0

21 failures, 29 successes (success rate: 58.00%)
2374 successful controllers (mean: 47.48)
```

A.1.4 Continuously Emitting, Mobile Radar

```
case-2003-11-07-1113-01      6
case-2003-11-07-1113-02      5
case-2003-11-07-1113-03     25
case-2003-11-07-1113-04    125
case-2003-11-07-1113-05    179
case-2003-11-07-1113-06     98
case-2003-11-07-1113-07     32
case-2003-11-07-1113-08    110
case-2003-11-07-1113-09     30
case-2003-11-07-1113-10    206
case-2003-11-07-1113-11      0
case-2003-11-07-1113-12     75
case-2003-11-07-1113-13      3
case-2003-11-07-1113-14     84
case-2003-11-07-1113-15     19
case-2003-11-07-1113-16      0
case-2003-11-07-1113-17      2
case-2003-11-07-1113-18      0
```

```
case-2003-11-07-1113-19      0
case-2003-11-07-1113-20     72
case-2003-11-07-1113-21      2
case-2003-11-07-1113-22      0
case-2003-11-07-1113-23    135
case-2003-11-07-1113-24      0
case-2003-11-07-1113-25      0
case-2003-11-07-1113-26     86
case-2003-11-07-1113-27     26
case-2003-11-07-1113-28     50
case-2003-11-07-1113-29     83
case-2003-11-07-1113-30      1
case-2003-11-07-1113-31      1
case-2003-11-07-1113-32      0
case-2003-11-07-1113-33      0
case-2003-11-07-1113-34      0
case-2003-11-07-1113-35      0
case-2003-11-07-1113-36     37
case-2003-11-07-1113-37      0
case-2003-11-07-1113-38     24
case-2003-11-07-1113-39    130
case-2003-11-07-1113-40     30
case-2003-11-07-1113-41    131
case-2003-11-07-1113-42      0
case-2003-11-07-1113-43     26
case-2003-11-07-1113-44    143
case-2003-11-07-1113-45    156
case-2003-11-07-1113-46      1
case-2003-11-07-1113-47      1
case-2003-11-07-1113-48    123
case-2003-11-07-1113-49      9
case-2003-11-07-1113-50      0

14 failures, 36 successes (success rate: 72.00%)
2266 successful controllers (mean: 45.32)
```

A.1.5 Intermittently Emitting, Mobile Radar with Regular Period

```
case-2003-12-08-1039-01      0
case-2003-12-08-1039-02      0
case-2003-12-08-1039-03     53
case-2003-12-08-1039-04      0
case-2003-12-08-1039-05      0
case-2003-12-08-1039-06      9
case-2003-12-08-1039-07      0
case-2003-12-08-1039-08      0
case-2003-12-08-1039-09      6
case-2003-12-08-1039-10      0
case-2003-12-08-1039-11     26
case-2003-12-08-1039-12      0
case-2003-12-08-1039-13      0
case-2003-12-08-1039-14      0
case-2003-12-08-1039-15      0
case-2003-12-08-1039-16     34
case-2003-12-08-1039-17     80
case-2003-12-08-1039-18      0
case-2003-12-08-1039-19     93
case-2003-12-08-1039-20      0
```

```
case-2003-12-08-1039-21     84
case-2003-12-08-1039-22      0
case-2003-12-08-1039-23      6
case-2003-12-08-1039-24     53
case-2003-12-08-1039-25      0
case-2003-12-08-1039-26      0
case-2003-12-08-1039-27      0
case-2003-12-08-1039-28      0
case-2003-12-08-1039-29      0
case-2003-12-08-1039-30      0
case-2003-12-08-1039-31      0
case-2003-12-08-1039-32      2
case-2003-12-08-1039-33      0
case-2003-12-08-1039-34      0
case-2003-12-08-1039-35      0
case-2003-12-08-1039-36     22
case-2003-12-08-1039-37     22
case-2003-12-08-1039-38      0
case-2003-12-08-1039-39     38
case-2003-12-08-1039-40      0
case-2003-12-08-1039-41      0
case-2003-12-08-1039-42      0
case-2003-12-08-1039-43     40
case-2003-12-08-1039-44      0
case-2003-12-08-1039-45      1
case-2003-12-08-1039-46      0
case-2003-12-08-1039-47      0
case-2003-12-08-1039-48      0
case-2003-12-08-1039-49      0
case-2003-12-08-1039-50      0

34 failures, 16 successes (success rate: 32.00%)
569 successful controllers (mean: 11.38)
```

A.2 Incremental Evolution

A.2.1 Seed Population

```
case-2003-12-19-1049-01-A      6
case-2003-12-19-1049-02-A     64
case-2003-12-19-1049-03-A     66
case-2003-12-19-1049-04-A    156
case-2003-12-19-1049-05-A     63
case-2003-12-19-1049-06-A     78
case-2003-12-19-1049-07-A     14
case-2003-12-19-1049-08-A      0
case-2003-12-19-1049-09-A     50
case-2003-12-19-1049-10-A     40
case-2003-12-19-1049-11-A     78
case-2003-12-19-1049-12-A     99
case-2003-12-19-1049-13-A     34
case-2003-12-19-1049-14-A     35
case-2003-12-19-1049-15-A    161
case-2003-12-19-1049-16-A     59
```

```
case-2003-12-19-1049-17-A     88
case-2003-12-19-1049-18-A     81
case-2003-12-19-1049-19-A     67
case-2003-12-19-1049-20-A     66
case-2003-12-19-1049-21-A    135
case-2003-12-19-1049-22-A     51
case-2003-12-19-1049-23-A     70
case-2003-12-19-1049-24-A     38
case-2003-12-19-1049-25-A     68
case-2003-12-19-1049-26-A     35
case-2003-12-19-1049-27-A      1
case-2003-12-19-1049-28-A    112
case-2003-12-19-1049-29-A     56
case-2003-12-19-1049-30-A     45
case-2003-12-19-1049-31-A     68
case-2003-12-19-1049-32-A     58
case-2003-12-19-1049-33-A      0
case-2003-12-19-1049-34-A     79
case-2003-12-19-1049-35-A      0
case-2003-12-19-1049-36-A     35
case-2003-12-19-1049-37-A     61
case-2003-12-19-1049-38-A     36
case-2003-12-19-1049-39-A      5
case-2003-12-19-1049-40-A     27
case-2003-12-19-1049-41-A      0
case-2003-12-19-1049-42-A     18
case-2003-12-19-1049-43-A    129
case-2003-12-19-1049-44-A     34
case-2003-12-19-1049-45-A     54
case-2003-12-19-1049-46-A     44
case-2003-12-19-1049-47-A     79
case-2003-12-19-1049-48-A    166
case-2003-12-19-1049-49-A      0
case-2003-12-19-1049-50-A      6

5 failures, 45 successes (success rate: 90.00%)
2815 successful controllers (mean: 56.30)
```

A.2.2 Intermittently Emitting, Stationary Radar

```
case-2003-12-19-1049-01-B      0
case-2003-12-19-1049-02-B     56
case-2003-12-19-1049-03-B    134
case-2003-12-19-1049-04-B     90
case-2003-12-19-1049-05-B      0
case-2003-12-19-1049-06-B     26
case-2003-12-19-1049-07-B     44
case-2003-12-19-1049-08-B      0
case-2003-12-19-1049-09-B     54
case-2003-12-19-1049-10-B      0
case-2003-12-19-1049-11-B    100
case-2003-12-19-1049-12-B     13
case-2003-12-19-1049-13-B    126
case-2003-12-19-1049-14-B     48
case-2003-12-19-1049-15-B    174
case-2003-12-19-1049-16-B     46
case-2003-12-19-1049-17-B     23
case-2003-12-19-1049-18-B     73
```

```
case-2003-12-19-1049-19-B     43
case-2003-12-19-1049-20-B    104
case-2003-12-19-1049-21-B     78
case-2003-12-19-1049-22-B     90
case-2003-12-19-1049-23-B     87
case-2003-12-19-1049-24-B      0
case-2003-12-19-1049-25-B      1
case-2003-12-19-1049-26-B     44
case-2003-12-19-1049-27-B     13
case-2003-12-19-1049-28-B     24
case-2003-12-19-1049-29-B      0
case-2003-12-19-1049-30-B      0
case-2003-12-19-1049-31-B    165
case-2003-12-19-1049-32-B      0
case-2003-12-19-1049-33-B      0
case-2003-12-19-1049-34-B    105
case-2003-12-19-1049-35-B      0
case-2003-12-19-1049-36-B      0
case-2003-12-19-1049-37-B     91
case-2003-12-19-1049-38-B    108
case-2003-12-19-1049-39-B      0
case-2003-12-19-1049-40-B    104
case-2003-12-19-1049-41-B      0
case-2003-12-19-1049-42-B      1
case-2003-12-19-1049-43-B     10
case-2003-12-19-1049-44-B      0
case-2003-12-19-1049-45-B      0
case-2003-12-19-1049-46-B      6
case-2003-12-19-1049-47-B    125
case-2003-12-19-1049-48-B    136
case-2003-12-19-1049-49-B    184
case-2003-12-19-1049-50-B      0

16 failures, 34 successes (success rate: 68.00%)
2526 successful controllers (mean: 50.52)
```

A.2.3 Continuously Emitting, Mobile Radar

```
case-2004-02-02-1012-01-B      0
case-2004-02-02-1012-02-B    115
case-2004-02-02-1012-03-B     54
case-2004-02-02-1012-04-B     93
case-2004-02-02-1012-05-B     16
case-2004-02-02-1012-06-B    139
case-2004-02-02-1012-07-B      7
case-2004-02-02-1012-08-B      0
case-2004-02-02-1012-09-B     81
case-2004-02-02-1012-10-B      1
case-2004-02-02-1012-11-B    109
case-2004-02-02-1012-12-B    127
case-2004-02-02-1012-13-B     45
case-2004-02-02-1012-14-B     79
case-2004-02-02-1012-15-B    179
case-2004-02-02-1012-16-B     59
case-2004-02-02-1012-17-B     60
case-2004-02-02-1012-18-B    105
case-2004-02-02-1012-19-B     33
case-2004-02-02-1012-20-B     31
```

```
case-2004-02-02-1012-21-B     78
case-2004-02-02-1012-22-B    139
case-2004-02-02-1012-23-B     63
case-2004-02-02-1012-24-B      8
case-2004-02-02-1012-25-B     56
case-2004-02-02-1012-26-B     28
case-2004-02-02-1012-27-B      0
case-2004-02-02-1012-28-B     15
case-2004-02-02-1012-29-B     16
case-2004-02-02-1012-30-B     16
case-2004-02-02-1012-31-B     84
case-2004-02-02-1012-32-B     21
case-2004-02-02-1012-33-B     37
case-2004-02-02-1012-34-B     93
case-2004-02-02-1012-35-B      0
case-2004-02-02-1012-36-B    153
case-2004-02-02-1012-37-B     68
case-2004-02-02-1012-38-B     89
case-2004-02-02-1012-39-B     30
case-2004-02-02-1012-40-B     24
case-2004-02-02-1012-41-B     16
case-2004-02-02-1012-42-B     29
case-2004-02-02-1012-43-B     82
case-2004-02-02-1012-44-B      3
case-2004-02-02-1012-45-B      0
case-2004-02-02-1012-46-B      6
case-2004-02-02-1012-47-B    118
case-2004-02-02-1012-48-B    152
case-2004-02-02-1012-49-B     12
case-2004-02-02-1012-50-B      5

5 failures, 45 successes (success rate: 90.00%)
2774 successful controllers (mean: 55.48)
```

A.2.4 Intermittently Emitting, Stationary Radar with Multiple Increments

```
case-2004-02-10-1428-01-C      0
case-2004-02-10-1428-02-C     47
case-2004-02-10-1428-03-C     13
case-2004-02-10-1428-04-C     81
case-2004-02-10-1428-05-C     10
case-2004-02-10-1428-06-C    138
case-2004-02-10-1428-07-C     24
case-2004-02-10-1428-08-C     12
case-2004-02-10-1428-09-C    143
case-2004-02-10-1428-10-C      0
case-2004-02-10-1428-11-C     55
case-2004-02-10-1428-12-C     56
case-2004-02-10-1428-13-C     27
case-2004-02-10-1428-14-C     61
case-2004-02-10-1428-15-C     23
case-2004-02-10-1428-16-C     19
case-2004-02-10-1428-17-C     47
case-2004-02-10-1428-18-C    117
case-2004-02-10-1428-19-C    109
case-2004-02-10-1428-20-C     68
case-2004-02-10-1428-21-C     24
case-2004-02-10-1428-22-C     47
```

```
case-2004-02-10-1428-23-C     26
case-2004-02-10-1428-24-C     19
case-2004-02-10-1428-25-C     22
case-2004-02-10-1428-26-C      0
case-2004-02-10-1428-27-C     14
case-2004-02-10-1428-28-C    105
case-2004-02-10-1428-29-C      0
case-2004-02-10-1428-30-C     22
case-2004-02-10-1428-31-C     80
case-2004-02-10-1428-32-C      0
case-2004-02-10-1428-33-C     30
case-2004-02-10-1428-34-C     88
case-2004-02-10-1428-35-C      0
case-2004-02-10-1428-36-C    139
case-2004-02-10-1428-37-C     39
case-2004-02-10-1428-38-C     26
case-2004-02-10-1428-39-C      1
case-2004-02-10-1428-40-C     10
case-2004-02-10-1428-41-C     55
case-2004-02-10-1428-42-C     24
case-2004-02-10-1428-43-C     43
case-2004-02-10-1428-44-C      8
case-2004-02-10-1428-45-C      0
case-2004-02-10-1428-46-C      0
case-2004-02-10-1428-47-C     52
case-2004-02-10-1428-48-C    122
case-2004-02-10-1428-49-C     24
case-2004-02-10-1428-50-C     13

8 failures, 42 successes (success rate: 84.00%)
2083 successful controllers (mean: 41.66)
```

A.2.5 Intermittently Emitting, Mobile Radar with Multiple Increments

```
case-2004-02-18-1108-01-D      0
case-2004-02-18-1108-02-D     62
case-2004-02-18-1108-03-D      3
case-2004-02-18-1108-04-D     69
case-2004-02-18-1108-05-D      8
case-2004-02-18-1108-06-D     59
case-2004-02-18-1108-07-D     90
case-2004-02-18-1108-08-D      0
case-2004-02-18-1108-09-D     18
case-2004-02-18-1108-10-D      0
case-2004-02-18-1108-11-D     62
case-2004-02-18-1108-12-D      6
case-2004-02-18-1108-13-D     47
case-2004-02-18-1108-14-D     10
case-2004-02-18-1108-15-D    100
case-2004-02-18-1108-16-D      2
case-2004-02-18-1108-17-D     41
case-2004-02-18-1108-18-D     79
case-2004-02-18-1108-19-D     30
case-2004-02-18-1108-20-D      7
case-2004-02-18-1108-21-D      0
case-2004-02-18-1108-22-D     51
case-2004-02-18-1108-23-D     46
case-2004-02-18-1108-24-D     40
```

```
case-2004-02-18-1108-25-D     11
case-2004-02-18-1108-26-D     19
case-2004-02-18-1108-27-D      0
case-2004-02-18-1108-28-D     14
case-2004-02-18-1108-29-D      0
case-2004-02-18-1108-30-D      0
case-2004-02-18-1108-31-D    105
case-2004-02-18-1108-32-D      0
case-2004-02-18-1108-33-D     64
case-2004-02-18-1108-34-D     60
case-2004-02-18-1108-35-D      0
case-2004-02-18-1108-36-D    123
case-2004-02-18-1108-37-D     11
case-2004-02-18-1108-38-D     34
case-2004-02-18-1108-39-D      0
case-2004-02-18-1108-40-D     48
case-2004-02-18-1108-41-D      3
case-2004-02-18-1108-42-D     19
case-2004-02-18-1108-43-D      0
case-2004-02-18-1108-44-D     12
case-2004-02-18-1108-45-D      0
case-2004-02-18-1108-46-D     61
case-2004-02-18-1108-47-D    143
case-2004-02-18-1108-48-D     39
case-2004-02-18-1108-49-D      6
case-2004-02-18-1108-50-D      0

13 failures, 37 successes (success rate: 74.00%)
1602 successful controllers (mean: 32.04)
```

Appendix B

Sample Results from Evolutionary Runs

B.1 Continuously Emitting, Stationary Radar

The results from one of the evolutionary runs described in Section 5.3.1 are shown. The evolutionary run was *case-2003-10-10-1313-10*. Successful individuals are marked (−>).

	Normalized Dist	Circling Dist	Level Time	Turn Cost
	0.328820	3.543143	4070.666810	0.000000
->	0.043496	2.027485	3690.166630	0.044907
	0.313649	13.205677	6930.766660	0.000056
	0.310075	25.628654	7200.000000	0.000000
	0.301030	3.518343	4716.600040	0.002472
	0.046516	1.578667	552.766720	0.175590
	0.277222	9.667850	7020.499990	0.000139
	0.303224	2.151518	4289.366760	0.050394
	0.316156	4.130311	4808.300020	0.002056
	0.304225	11.200435	6905.200000	0.000056
	0.238432	2.549245	3841.133420	0.001870
	0.239488	2.317055	4194.933470	0.010602
	0.060962	1.370775	0.000000	0.296808
	0.319152	3.752388	5085.200040	0.011481
	0.299405	3.539395	5167.266690	0.012037
	0.256112	1.789918	2872.866520	0.823625
	0.223748	2.518139	3844.266660	0.002593
	0.058804	1.506111	0.433350	0.092806
	0.320791	3.892854	4891.799930	0.002028
	0.274823	2.436951	3234.400020	0.000056

	0.267754	3.344972	4382.099910	0.002463
	0.291799	2.270008	3860.000000	0.004667
	0.269604	3.001440	5047.233280	0.011620
	0.324462	3.746637	4942.033390	0.002213
	0.306368	3.086205	4293.833310	0.002444
	0.321781	6.025094	5131.166690	0.000028
	0.257992	2.391825	4079.566650	0.002639
	0.241385	4.334878	4660.366670	0.002208
	0.236384	3.228717	4264.100040	0.002398
	0.282776	3.665463	4527.333370	0.002065
	0.290233	5.989334	4925.966640	0.000037
	0.283623	5.758885	5290.433350	0.001903
	0.260676	4.258947	4581.633300	0.010972
	0.188623	10.873907	6154.400020	0.000167
	0.325985	4.904173	5325.033260	0.001917
	0.206922	17.458660	6466.500020	0.000120
	0.232861	5.522527	5038.833310	0.002306
	0.271703	3.648474	4792.700040	0.010417
	0.182618	5.172512	4605.833440	0.001935
	0.218467	2.231208	2701.300050	0.000046
	0.230024	4.586133	4451.666560	0.002639
	0.180701	7.093325	5445.099950	0.000056
	0.272630	2.606339	4455.566710	0.010556
	0.234449	3.777302	4815.266720	0.011157
	0.201870	29.314140	7200.000000	0.090951
	0.326543	2.269419	3818.299870	0.004421
	0.208220	1.875763	3134.333190	0.045463
	0.323054	3.517340	4556.433410	0.002426
	0.284905	3.244854	4062.366640	0.001731
	0.264632	2.313873	2824.266660	0.000046
	0.177821	7.559918	5575.200040	0.000083
	0.287924	2.940615	4504.233400	0.008676
	0.200530	9.728267	5998.799970	0.000056
->	0.094156	1.812824	2554.733280	0.043472
	0.231354	4.105953	4481.566770	0.001889
	0.289041	7.971835	6123.666690	0.000083
	0.259654	9.857268	6721.966670	0.000037
	0.265673	1.971685	2778.533330	0.003167
	0.196898	2.309340	3674.133300	0.009931
	0.244472	5.341781	4146.799930	0.000028
	0.222073	2.249782	3544.500120	0.002491
	0.154483	5.259242	3537.966610	0.000028
	0.278709	3.101660	4180.400090	0.002083
	0.296840	3.736434	4658.733370	0.002361
	0.153186	5.854900	4171.366580	0.000028
	0.193374	2.045116	2316.499940	0.002611
->	0.120276	2.106473	2021.766970	0.000000
	0.219799	10.621859	6237.666700	0.000111
	0.316809	9.504509	6488.766630	0.000111
	0.079355	1.473197	1137.800290	1.436192
	0.298255	4.880388	4960.599980	0.001944
	0.261762	2.362818	4589.599910	0.010995
->	0.092876	2.060139	2653.333440	0.009505
	0.224780	9.858183	6489.100040	0.000000
	0.279769	8.791500	6234.599990	0.000028
	0.281483	2.531167	4347.699890	0.010394
	0.175818	2.866533	3428.200070	0.000000
	0.190072	2.441378	3379.433290	0.002333
	0.192680	2.022956	2575.466610	0.004667
	0.270622	2.943931	3609.200130	0.000102
	0.214735	1.884999	3053.566590	0.040694
	0.195780	1.905717	3258.233340	0.043194
	0.253177	2.282519	4348.099980	0.011065
	0.249192	11.981641	6910.066660	0.000083
	0.252003	6.052683	4780.299990	0.000000
	0.078552	1.483501	1157.566530	1.368289

->	0.146897	2.068794	1784.666750	0.000000
	0.191846	1.972041	3460.499880	0.046481
	0.179657	15.666398	6435.266650	0.000185
	0.278084	2.033293	3704.266660	0.045625
	0.110042	5.866265	3935.633240	0.000028
	0.162199	5.580391	3959.366760	0.000056
->	0.090710	1.784386	2350.299990	0.039537
	0.183969	2.863065	3110.533450	0.000000
	0.254154	2.225771	4210.466610	0.012454
->	0.099449	1.836767	2862.566530	0.048981
	0.243109	2.396750	4020.799870	0.008898
	0.297805	2.835974	4252.799990	0.002718
	0.323636	2.098077	4208.166810	0.048171
	0.225324	2.547789	4267.466740	0.012222
	0.273659	3.207853	4452.766720	0.002116
	0.141800	1.579565	1996.233520	0.798569
	0.242260	2.163006	4030.166630	0.013194
	0.172593	23.851023	6922.733330	0.132113
	0.318003	4.602868	5089.299930	0.002306
	0.210684	7.544898	5921.233370	0.000167
->	0.109155	1.805050	2920.499880	0.040648
	0.048246	1.594425	399.666750	0.059350
	0.243677	36.932926	7200.000000	0.015616
	0.052121	1.581823	445.766600	0.059889
	0.208793	7.302366	5712.033390	0.000111
	0.072926	1.442787	1049.066770	1.449736
	0.066020	1.386420	879.733280	1.042463
	0.167870	1.908485	2970.066530	0.038935
->	0.146090	2.291370	3673.433230	0.009120
	0.293331	10.220240	6941.066670	0.000083
	0.165084	2.137643	2184.666750	0.000000
	0.077853	1.423778	1070.333250	1.092616
	0.327857	2.181463	4467.133480	0.049514
	0.138658	11.297136	5777.100070	0.000000
	0.228153	2.133630	3887.733460	0.043843
->	0.088123	2.014827	2583.733220	0.008241
->	0.071040	1.686669	1168.366700	0.041523
	0.140881	6.746524	5077.866670	0.000111
	0.258950	1.993392	3755.233460	0.047083
	0.227561	2.222646	2679.866640	0.000000
	0.181616	2.067945	3705.166630	0.011481
	0.071843	1.480278	1004.633180	1.499153
	0.080165	1.590129	1145.700070	0.692074
->	0.144676	2.328420	3293.933410	0.009722
	0.052873	1.607855	648.699950	0.197396
	0.287231	20.099016	7116.333330	0.000139
	0.251144	2.174154	4225.700070	0.050185
	0.161347	1.932470	3451.000060	0.042685
	0.124289	1.652987	1334.066770	0.583449
	0.212667	2.409988	4078.200070	0.008981
	0.285461	3.476030	4072.766720	0.001833
	0.075567	1.438212	1133.066410	1.336458
	0.068490	1.368270	950.866700	0.997535
	0.142272	9.163619	5323.800050	0.000028
->	0.135120	1.739015	2376.700130	0.048056
->	0.085127	1.729598	2168.466800	0.040324
->	0.137580	2.060590	2744.866640	0.008472
	0.057187	1.613153	732.566530	0.208241
->	0.115988	1.919777	1762.500000	0.003861
	0.170906	2.585834	3512.533260	0.002194
	0.065161	1.559165	0.633540	0.105035
->	0.123545	2.796618	2450.266720	0.000093
	0.194325	2.154107	3424.833370	0.009329
	0.175257	1.909556	3185.466610	0.047222
	0.168702	5.857605	4374.033200	0.000028
	0.215346	2.129191	3522.466740	0.002671

	0.063171	1.409424	858.400270	1.316169
	0.206156	2.262801	4015.700070	0.009167
	0.076409	1.499340	1133.900150	1.328637
	0.286158	7.467243	5857.899930	0.000028
	0.317776	3.558345	4537.166750	0.000000
	0.151334	1.921682	3245.933230	0.041620
	0.289663	5.991089	5332.200010	0.002042
	0.062559	1.334128	832.100220	0.968877
	0.166988	1.908238	3100.033260	0.046065
->	0.123074	2.056466	1762.800290	0.000000
->	0.133514	2.809313	3373.233340	0.002056
->	0.104534	1.798739	2558.533330	0.042847
	0.252646	2.891489	3848.900150	0.001639
	0.074673	1.464298	1060.999760	1.382817
	0.226185	1.649907	2341.533200	0.707319
	0.178724	2.064924	3043.533330	0.002407
	0.266344	2.422918	4324.333190	0.010856
	0.096010	5.593229	3524.766540	0.000213
->	0.135918	1.920369	3059.866640	0.045023
	0.118618	5.552805	3772.666630	0.000028
	0.049359	1.862902	0.000000	0.000102
	0.209574	5.777194	4909.666600	0.000111
	0.096741	1.566996	914.866940	0.542431
	0.295287	2.401360	5035.700070	0.009769
	0.121242	1.469103	1613.900150	1.368204
	0.245381	11.819967	7200.000000	0.000347
	0.114964	5.920982	4173.099980	0.000111
	0.114135	1.540771	1309.000240	0.579204
	0.066877	1.424705	917.833250	1.261718
	0.187969	15.073728	6787.000010	0.000139
	0.048756	1.605561	588.966670	0.183687
	0.166184	3.254321	3635.966800	0.002120
	0.217830	1.781222	2535.433350	0.040741
	0.209940	6.435883	5367.933350	0.000000
	0.152148	11.015396	5496.133270	0.000111
->	0.107103	2.234087	3000.766600	0.009190
	0.127788	5.265575	3152.066650	0.000000
	0.165875	2.127235	2176.766660	0.000000
->	0.089181	2.141595	1693.066410	0.000000
	0.055922	1.632667	694.066770	0.209829
	0.185083	2.734225	3940.400090	0.002306
	0.176794	8.754511	5809.433290	0.000028
->	0.092294	1.740000	2417.933350	0.042741
	0.057949	1.636603	910.399780	0.034685
	0.171710	1.894420	2967.999880	0.039120
	0.077234	1.472978	1102.266850	1.301410
	0.139551	1.599112	2141.633300	0.716509
	0.072324	1.389745	1012.366940	1.097461
	0.232398	1.996413	3636.266780	0.043912
	0.143046	1.634357	2160.266720	0.779139
->	0.111555	2.658652	2264.233400	0.000000
->	0.138236	2.544327	3210.833440	0.004000
->	0.101203	2.171790	2778.833310	0.002333
->	0.122369	2.744834	3103.299870	0.002935
	0.286657	9.676486	6691.633340	0.000083
	0.213822	7.299954	5810.099950	0.000083
->	0.087559	1.911703	2098.866580	0.004005
	0.156819	11.710814	5780.599980	0.000083
->	0.149904	2.272169	3069.766540	0.002269
	0.226863	3.588670	4093.066710	0.000000
->	0.127110	1.808159	2525.166630	0.039398
->	0.102791	2.738637	2005.233150	0.000000
->	0.088869	1.737977	2165.233460	0.034611
	0.073674	1.596617	1122.633060	1.464984
->	0.110632	2.516468	2897.833250	0.002611
	0.084413	1.749138	0.033570	0.005713

	0.150468	1.954050	3174.700010	0.044074
->	0.101906	1.956894	2803.200070	0.045046
	0.185790	6.812111	5080.566710	0.000028
->	0.098730	1.920210	2733.066710	0.009815
	0.247918	6.199330	5393.699950	0.000389
	0.173090	3.847470	3913.833310	0.002000
	0.231781	2.120585	2201.400150	0.000208
	0.294663	3.104473	4422.399900	0.002319
	0.197490	14.641774	6591.166650	0.009949
->	0.085649	2.895359	2360.533450	0.002139
	0.134677	8.986752	5563.166660	0.000204
	0.203166	2.143956	3920.466610	0.008958
	0.296140	7.813691	6461.900020	0.000111
->	0.131972	1.837751	2694.633480	0.038519
	0.190954	4.009045	4255.700070	0.001944
	0.051514	1.623353	624.500120	0.193303
->	0.112177	1.865415	2909.033200	0.034606
	0.176913	7.079608	5427.299960	0.000139
->	0.132885	1.936081	3294.166560	0.049500
	0.115262	8.508290	4439.066770	0.000000
	0.061682	1.919829	0.000000	0.000000
	0.211313	12.597135	6262.966690	0.000074
	0.119602	5.467021	3606.433410	0.000028
	0.270179	2.188156	4135.366520	0.010185
->	0.148953	1.943068	3368.366700	0.047176
->	0.137002	2.706971	2723.299870	0.000056
	0.080638	1.444545	468.133540	0.475155
	0.156214	9.301848	5631.266630	0.000083
	0.055335	1.590526	467.466430	0.066826
	0.197940	7.396869	5661.933290	0.000111
	0.274031	6.239582	5680.500030	0.000139
	0.181990	1.802356	2407.399900	0.039491
	0.184696	2.770066	4011.266780	0.010046
	0.235698	3.014919	4432.866520	0.008954
	0.073984	1.342833	991.666870	2.618715
	0.263848	3.125758	4628.033450	0.009519
	0.213146	2.312295	2624.733280	0.000000
	0.143352	8.104529	4733.566740	0.000028
	0.064315	1.932548	0.033570	0.000000
->	0.063760	1.672294	1074.000240	0.039002
	0.163473	2.121502	3189.633480	0.004083
	0.119254	7.306860	5093.500060	0.000056
->	0.130578	2.157886	2197.233280	0.000000
	0.229393	2.308475	4102.933350	0.009630
->	0.126726	1.805576	2497.666630	0.040056
	0.170282	2.282141	3764.033200	0.010769
	0.262945	3.324694	4265.599980	0.002042
->	0.095573	2.778254	2779.433290	0.002389
	0.159104	11.595980	6148.799970	0.000083
	0.220851	2.781375	3687.200010	0.002083
	0.187508	10.155241	6102.699970	0.000514
->	0.087055	2.004236	1424.433590	0.000000
->	0.100663	2.091127	2135.566710	0.002676
	0.173637	9.283154	5711.433260	0.000083
->	0.129865	2.476919	3037.433470	0.002222
->	0.097593	1.916116	2613.999940	0.037546
	0.328758	3.743015	4850.066680	0.004199
	0.211890	2.168441	2400.700070	0.000194
	0.249939	15.920287	6981.866660	0.000093
	0.258363	19.484579	6551.399990	0.000046
->	0.147583	2.858304	3081.233220	0.000000
	0.128212	6.156535	4202.366640	0.000000
	0.198495	5.971496	3933.200070	0.000000
	0.105664	1.490751	1375.733030	0.719671
->	0.149401	2.529089	3414.599910	0.008981
	0.047393	1.590574	394.866940	0.057586

	0.107710	6.538826	4779.900050	0.000083
	0.255273	2.060595	4110.966800	0.011343
	0.154874	2.620379	3575.666810	0.002444
	0.086168	1.528718	1212.933350	0.609167
	0.169931	2.237469	3149.200130	0.002477
	0.205725	2.395374	3750.766600	0.004662
->	0.106257	1.760780	2211.866760	0.039954
	0.155429	2.605528	3841.199950	0.009676
	0.051093	1.608321	430.200200	0.062447
	0.050580	1.278418	636.433110	1.049324
	0.198893	1.987814	3504.933470	0.043704
	0.307000	11.482088	7099.833340	0.001083
	0.235209	2.567712	4365.533450	0.009676
	0.103870	6.894420	4876.466670	0.000056
	0.111098	1.529349	1615.966800	0.767579
	0.164014	1.875882	3001.300050	0.047824
	0.113336	10.188980	4170.433350	0.000000
->	0.095039	2.189666	2511.366580	0.003157
	0.294035	2.385769	3950.366520	0.008148
->	0.132488	2.224407	2787.999880	0.002333
	0.267153	2.631441	3907.533260	0.002356
	0.292680	2.582195	4152.900090	0.002852
	0.070548	1.376981	959.933470	1.073012
->	0.126134	2.027290	3184.633480	0.009028
->	0.103609	1.785167	2351.933290	0.041528
	0.263635	2.355022	4291.600040	0.010338
	0.262272	2.876414	5067.299960	0.012935
	0.327859	13.366334	6697.966650	0.000028
	0.205214	2.196081	2509.933470	0.000088
->	0.075157	1.716776	1297.433470	0.043685
	0.108052	1.561376	1829.899900	0.712130
	0.228873	2.437615	3676.600040	0.002056
	0.191187	2.362020	2919.200130	0.000000
	0.204361	5.438784	3933.699950	0.000028
	0.217356	2.979040	3922.466740	0.000000
->	0.086594	1.868031	1707.333370	0.005222
	0.145407	6.493283	4940.200040	0.000139
	0.124803	5.185487	3494.299930	0.000028
->	0.094656	2.529356	1996.266480	0.000028
	0.190158	17.369381	6988.866670	0.211181
	0.167364	6.042999	5013.566740	0.000056
	0.061986	1.355383	331.933590	0.368528
	0.047296	1.278870	203.099980	0.363042
	0.216941	1.984444	3516.799930	0.046852
	0.241720	6.324976	5170.166630	0.000056
->	0.129078	2.023129	3283.566590	0.008556
	0.187003	2.138104	2960.400090	0.002370
	0.202428	2.233839	2686.300050	0.000093
	0.128742	8.936490	5195.866700	0.000111
	0.150954	2.970555	3094.033200	0.000139
	0.200083	1.858249	2929.233400	0.045509
	0.053958	1.609353	465.833130	0.066431
	0.308103	3.371641	5646.166690	0.012083
	0.067880	1.698506	1339.866940	0.253972
	0.065534	1.379713	851.799930	1.093567
	0.160936	2.039658	3463.733220	0.011065
	0.081176	1.537926	1196.433110	1.596155
	0.169340	2.148153	3690.000000	0.010926
	0.058280	1.900315	0.000000	0.000028
	0.204825	6.062325	4356.266780	0.000000
	0.069128	1.381920	396.733400	0.408713
	0.295808	4.261728	4896.566620	0.001861
	0.246178	6.563236	5379.499970	0.000000
	0.097155	1.522639	1326.533200	0.580676
	0.308644	3.703122	5081.199950	0.009815
	0.158610	2.158397	3566.400150	0.009167

	0.083006	1.465377	1188.499760	1.171729
	0.070129	1.385426	967.700200	1.072340
->	0.106606	2.361647	2841.366580	0.008843
	0.280271	2.140215	4165.400090	0.012130
	0.309264	2.391415	3965.666810	0.002620
	0.162824	2.633784	3751.333310	0.009398
->	0.076764	1.724398	1356.166380	0.045808
->	0.091468	1.823706	2357.466740	0.040046
->	0.084074	2.073960	2537.833250	0.008657
->	0.082604	2.911652	2453.066710	0.002778
->	0.089968	2.287374	2564.766540	0.003167
->	0.102965	2.430966	2568.800050	0.000370
	0.174605	5.072166	4686.699980	0.004046
	0.215939	3.057092	4153.500060	0.002653
	0.309479	3.639834	4264.599910	0.000000
	0.266841	8.991049	6157.833330	0.000083
->	0.098117	2.655870	2997.600100	0.008611
->	0.118007	2.606844	3091.666560	0.003546
	0.158255	10.106637	5707.799990	0.000028
	0.174052	9.124874	5652.366640	0.000194
	0.171501	7.661305	5419.666600	0.000056
	0.056430	1.647215	709.266970	0.208646
	0.157282	2.170696	3146.233220	0.002755
	0.049913	1.563288	389.033200	0.060051
->	0.113847	2.622050	2928.900150	0.002583
	0.245613	3.605893	4509.500120	0.011690
->	0.117288	1.926753	2170.633240	0.004083
	0.248432	2.682995	4628.566590	0.012130
	0.274663	8.248817	6118.300020	0.000056
->	0.100170	1.935462	2475.433350	0.008750
->	0.129616	2.371081	3017.466740	0.002130
	0.069700	1.652285	1085.300290	0.285741
	0.155810	2.282239	2489.766540	0.002130
	0.216481	7.590241	5657.833400	0.000056
	0.144043	7.124513	5332.400050	0.000083
	0.199426	1.909960	3435.266720	0.044630
	0.139745	6.611647	5062.766720	0.000139
	0.159579	2.020190	2752.166750	0.009981
	0.131458	5.705298	4038.866580	0.000028
->	0.125173	1.842356	2622.266540	0.036852
->	0.105045	2.095453	1523.066410	0.000000
	0.246736	2.018926	3456.766660	0.010463
	0.203773	1.900150	3072.066650	0.041528
	0.140325	6.900012	4897.566680	0.000083
	0.234772	2.078118	3719.066770	0.054630
->	0.091468	1.823688	2357.333370	0.040093
	0.056724	1.607440	493.200070	0.067806
	0.112930	1.678895	1840.366820	0.698597
->	0.112582	1.790687	2589.233400	0.047431
->	0.067137	1.701929	1127.733150	0.039544
	0.160488	7.107420	5375.466610	0.000167
->	0.082182	1.739538	2134.933470	0.041019
	0.220339	2.314074	3010.566710	0.000028
	0.152828	2.557069	3336.799930	0.002069
	0.067629	1.405145	937.600100	1.126023
	0.221499	2.775584	3771.233220	0.002144
->	0.090001	1.754084	2268.666690	0.041157
	0.148385	4.981453	4664.799960	0.002074
	0.202738	8.672126	5860.233310	0.000167
->	0.116599	2.954058	2795.533450	0.000000
	0.184082	1.620524	1563.499760	0.619306
->	0.136567	1.734678	2233.099980	0.041111
->	0.101741	1.989125	1342.833250	0.000000
	0.280614	4.648739	5352.633360	0.010463
	0.054916	1.602092	667.800290	0.203507
	0.164553	2.760484	2912.799990	0.000000

->	0.117214	2.388905	2923.900150	0.002352
->	0.081856	2.401320	2192.799990	0.002630
	0.236209	4.761230	4575.833440	0.002389
	0.247354	2.381311	2922.966610	0.000000
	0.307553	26.691856	7200.000000	0.000028
	0.054409	1.596664	447.899780	0.065681
->	0.145561	2.264740	3526.333310	0.009722
	0.250151	23.218376	7200.000000	0.000042
	0.152443	3.947146	3845.400090	0.003218
	0.081494	1.511928	1222.333370	1.344282
	0.186637	6.215370	5658.866730	0.000083
	0.226497	3.590280	4107.333370	0.002417
	0.213663	2.356901	2646.533200	0.000000
	0.262664	2.683252	3857.366640	0.002306
	0.053392	1.619068	451.333620	0.066153
	0.050160	1.278336	584.533080	0.915623
	0.179212	10.834148	5715.899960	0.000000
	0.199668	1.472187	1574.466550	0.671060
	0.292711	2.286567	3410.433350	0.000000
->	0.136282	1.954420	2330.866700	0.003861
	0.163029	1.971523	3502.133480	0.047731
	0.221306	6.770026	5824.833370	0.000028
->	0.125575	1.893157	3058.566590	0.043069
	0.169075	2.252194	2621.066590	0.002250
	0.157804	5.063101	3685.299990	0.000000
	0.144128	6.766976	4554.266660	0.000028
	0.306489	2.437555	3495.033260	0.000000
->	0.147854	1.947944	3002.566530	0.010787
	0.294244	4.780453	4786.266630	0.002000
->	0.083588	2.111602	2503.066710	0.007870
->	0.130844	2.649191	2927.533260	0.000167
	0.105305	1.705421	1461.866460	0.675801
	0.247234	2.353060	4498.333440	0.012269
->	0.108604	1.778091	2442.433470	0.042639
->	0.134084	2.101109	3067.300110	0.008356
	0.195390	2.726004	3151.933290	0.000000
->	0.098402	1.727059	2092.933350	0.040648
	0.131198	1.628699	1458.233030	0.559565
	0.280968	2.263729	4410.366520	0.011250
	0.160185	5.535900	3841.366580	0.000028
	0.116483	1.513529	1683.066410	0.656236
	0.118094	8.518476	4914.966740	0.000083
	0.159877	3.144639	3934.166560	0.002435
->	0.121418	2.128010	2126.666560	0.000046
	0.203933	1.925577	3346.499940	0.040301
	0.250621	5.957766	4258.699950	0.000000
	0.254625	2.831851	3291.466670	0.000000
->	0.083452	1.983373	1062.266850	0.000000
->	0.092562	2.568306	2513.466800	0.002602
->	0.121833	2.124071	2083.533330	0.000000
	0.194839	6.911875	5183.133390	0.000056
	0.211707	2.929800	3800.599980	0.002111
	0.222048	3.411309	4014.166560	0.001944
	0.069343	1.448476	974.366460	1.302597
->	0.134061	2.103528	1806.799930	0.000000
	0.053546	1.368817	653.633420	1.335484
->	0.125819	2.138354	3101.499940	0.002481
	0.246465	2.014380	3448.766780	0.010926
->	0.140507	3.730721	3723.466800	0.002167
	0.064165	1.262791	827.733150	2.342995
	0.307419	4.003674	4401.733400	0.000000
	0.174442	2.300458	2427.166750	0.001458
	0.195165	15.495749	6226.800000	0.000093
->	0.108670	2.921218	2738.933410	0.002329
	0.216157	13.564490	6402.799990	0.000046
	0.308414	2.435696	4081.766660	0.003088

	0.220105	4.344788	3987.600100	0.000000
	0.254737	4.369454	4722.233280	0.002083
	0.164393	2.902336	3872.166750	0.002352
	0.255165	2.256413	3763.433230	0.003898
->	0.121997	1.770952	2500.033260	0.041477
	0.186132	1.786236	2334.599910	0.038843
->	0.148215	2.211723	2348.466800	0.000093
	0.250738	7.987927	5991.933360	0.000028
	0.157620	2.002423	2486.300050	0.004431
	0.229046	2.094451	3064.066770	0.004028
	0.281227	1.772417	2565.933230	0.625329
	0.194588	1.931347	3254.266660	0.042315
	0.248399	2.083916	3335.899960	0.003991
	0.054642	1.608830	683.733520	0.203565
	0.186323	1.601910	1832.666630	0.564616
->	0.100240	1.820829	2559.566650	0.044074
	0.178993	5.789515	4435.199890	0.000028
	0.329759	9.792069	7200.000000	0.000056

B.2 Intermittently Emitting, Mobile Radar

These are the results from one of the evolutionary runs described in Section **??**. The evolutionary run was *case-2004-02-18-1108-15-D*. Successful individuals are marked (−>).

	Normalized Dist	Circling Dist	Level Time	Turn Cost
->	0.062796	2.343945	1842.100220	0.029593
	0.214082	34.205872	7200.000000	0.000000
	0.076570	3.302818	2715.733340	32.010311
	20.027145	6.830411	7200.000000	0.114639
->	0.063831	2.355052	1523.800050	0.027201
	0.297943	6.098861	5709.033360	0.000000
	2.296083	2.857692	3066.766660	0.213556
->	0.075052	2.734460	1582.033080	0.027171
->	0.075100	2.698218	1703.499760	0.025898
	0.076610	6.069944	3255.700070	7.664083
	0.302084	7.570784	6907.233330	15.999040
	0.181147	32.176781	6862.500000	0.034648
	0.148495	7.585789	5681.566620	4.911968
	0.131665	6.780853	4721.033330	5.910549
	0.094504	5.431774	3791.366580	27.815166
	0.178577	29.506134	6771.433330	0.000000
	0.083430	3.209027	2968.200070	1.090778
	0.093374	9.293422	4254.933470	0.271750
	0.080142	3.055355	3116.466670	18.287130
->	0.073351	2.779958	2398.433230	0.033887
	0.189929	26.890627	6696.333350	0.000000
	0.131911	4.416654	4215.899960	0.000000
->	0.079183	3.553635	2190.033260	0.026711
	0.092260	4.115616	3150.566710	0.026882
	0.143494	11.232306	5128.699950	0.000185
	0.123276	3.497137	3820.266720	20.764973
	0.153763	5.028705	4635.133360	0.000208
	0.180660	5.209774	4757.466740	0.000000
->	0.072909	2.994869	1821.933590	0.028970
	0.286044	25.626274	6842.433320	0.000000
	0.084081	3.616164	3241.099850	18.686634
->	0.088742	2.784264	2329.766540	0.026088
	0.283084	6.392275	5754.533390	0.000000
	0.268195	15.287744	6739.100000	0.035255
	0.294720	8.889200	6529.333340	0.039139
	0.091166	6.682488	4184.566650	0.026632
	0.314841	4.904294	6365.166700	0.033065
->	0.075218	3.423527	2155.766600	0.026532
->	0.071799	3.015669	1788.733520	0.026264
	0.110011	4.096508	2821.266780	0.000134
->	0.076083	2.746397	2368.533330	0.030512
	0.162163	12.016426	5956.833340	0.046296
	0.069005	3.137994	2686.900020	31.163347
	0.093426	3.814959	3685.499880	23.154144
	0.290193	5.568099	6501.266630	0.038042
	0.222820	14.400504	6670.600010	0.033780
->	0.071473	3.284093	1872.299800	0.025789
	0.154232	12.870954	5549.166720	0.032847
	0.390041	4.334053	6131.800000	0.037343
	0.196979	3.606488	4095.633240	26.576374

	0.135607	3.950111	4208.166810	27.592751
	0.266146	4.730499	6335.266650	0.035627
	0.207208	23.320135	7181.700000	0.157611
	0.076131	3.539616	2488.966670	0.151525
	0.210033	5.003224	4663.999940	0.000120
	0.101722	4.880534	4134.833370	0.820944
	0.159422	3.668174	3590.799870	0.029174
	0.221934	26.278076	7019.699990	0.000144
	0.187278	4.279787	5099.900050	0.034954
	0.136961	4.874965	4847.866670	0.037359
	0.085591	5.114722	3215.133360	0.026646
	0.076975	4.504138	2849.833370	25.531561
->	0.073413	2.765371	2312.000120	0.033741
	0.118011	9.703473	4818.533330	5.844199
->	0.081706	3.004213	2520.033260	0.032949
	0.298347	3.731478	5489.633330	0.042301
->	0.087217	2.658624	2647.600100	0.032593
	0.157265	21.758945	6767.200010	0.153028
	0.250648	4.351111	5041.666720	0.032310
	0.264967	18.126720	7036.766660	0.045410
->	0.079268	2.608000	1915.999760	0.024241
	0.250983	6.784039	5981.666640	0.000282
	0.197072	23.236704	6822.533340	0.000208
	0.093395	3.727706	3627.333370	24.264030
	0.085830	3.800285	3370.966800	29.177841
	0.100700	11.008328	4547.033390	0.035454
->	0.100977	3.650596	3325.733340	0.028826
	0.093925	5.561399	3649.666750	0.043505
->	0.078630	3.406617	1937.700200	0.025153
	0.121179	6.310108	4160.366520	0.000000
	0.093698	4.124692	3529.400020	0.033278
	0.086686	3.855139	3463.033450	24.438061
	0.096764	4.554658	3894.733280	16.766430
->	0.072768	3.262966	2293.966670	0.034023
	0.105186	3.038821	3354.800110	30.232035
	0.166251	11.053700	5520.666660	0.024343
	0.089439	5.427857	3484.100040	0.024824
	0.163896	16.693291	5941.966630	0.000000
	0.117623	6.783916	4523.366700	0.025252
	0.154441	18.703705	6012.200010	0.000185
	0.261027	4.685683	5732.200010	0.033188
	0.235983	6.277143	5774.266660	0.034109
	0.098877	3.735565	3574.066770	0.800685
	0.435428	3.696289	5504.566650	0.034581
->	0.112109	2.645873	2541.566770	0.027894
->	0.075269	3.265632	1891.699830	0.024030
	0.094142	5.649949	3850.033260	0.026937
	0.333976	4.570414	5569.933320	0.000000
	0.168999	24.186157	6801.066670	0.163924
	0.082699	9.747924	3564.833370	6.971514
	0.092493	4.724902	3610.933230	0.047081
	0.097618	7.216063	3512.000120	0.000000
	0.147337	12.179194	5535.033260	0.000000
	0.129291	5.558990	4610.433350	21.194805
->	0.071086	2.675460	1553.533330	0.026090
	0.135135	17.945797	5468.999940	0.000190
	0.145999	12.798586	5458.933260	0.041310
	0.138514	8.103776	4902.333370	0.032421
	0.319141	24.655748	7008.233340	0.000000
	0.370813	4.310419	5873.166660	0.031093
->	0.071805	3.031048	1852.233280	0.028947
	0.302188	23.430326	6989.533330	0.000000
	0.299280	4.426123	4920.899960	0.000292
	0.163832	3.485074	3713.933410	0.029250
	0.177498	22.701588	6720.499990	0.053419
->	0.085086	2.845344	2419.633480	0.029201

	0.192015	15.647902	5639.900050	0.000120
->	0.074605	3.094558	2465.899960	0.031470
->	0.077275	3.892542	2567.500000	0.031789
->	0.096395	3.917679	3827.033390	0.026771
	0.422489	8.007991	6937.433340	0.025231
	0.069312	2.961613	2242.000120	27.097231
	0.311917	3.613388	4828.866730	0.030722
->	0.078882	2.573016	1757.866820	0.022252
	0.464442	6.837708	7095.433330	0.000000
	0.330896	7.376103	6622.866670	0.000181
	0.177341	17.250416	6289.333340	0.000000
->	0.064238	2.488128	1540.533450	0.024787
	0.094565	5.833634	4267.133480	0.026556
	0.090009	7.403148	4081.900020	6.426412
->	0.075132	2.932461	2115.599980	0.032338
	0.198468	7.913723	6067.533340	0.000000
	0.124765	7.489634	4695.633390	0.027125
	0.100883	5.175179	4045.400090	17.531746
	0.072240	3.565946	2760.400090	0.238873
	0.224224	5.194539	4865.399930	0.000000
->	0.074077	3.851790	2354.666750	0.032644
	0.167147	14.963200	6058.466640	0.035949
	0.211560	6.887725	6376.299970	0.000000
	0.156320	5.081725	4598.066710	0.000000
	0.084849	4.136893	2889.800110	0.000000
	1.154532	9.257181	7116.800000	0.000208
	0.078305	3.111092	2951.766660	23.550722
	0.097487	4.854428	3544.866640	0.000185
	0.095756	6.229539	4111.566770	0.024262
	0.401579	13.824764	7163.166670	0.000144
	0.084810	5.700747	3307.333370	0.000000
	0.115286	9.995122	5180.333400	0.039861
	0.151224	13.466304	6026.133350	0.040609
	0.113500	5.279205	4413.599850	0.000208
	0.124605	8.718957	5063.500060	0.000000
	0.100542	8.003244	4360.100100	0.022208
	0.310429	4.606116	6278.266680	0.038377
	0.328600	5.191656	5906.933290	0.030623
	0.114648	12.032801	4891.333310	0.033947
	0.272296	5.555336	5083.033290	0.000000
	0.584188	3.699486	2026.966550	0.006391
	0.079569	4.032666	2734.866640	0.000148
->	0.089496	3.547662	2279.233400	0.022259
	0.233163	6.659484	5623.233340	0.000000
->	0.086869	3.076712	2628.266600	0.031403
	0.174040	19.278118	6982.066670	0.165296
	0.156903	5.619088	4875.866700	0.000167
	0.212167	19.570333	6661.266670	0.032449
	0.255794	4.512806	6185.299990	0.034891
->	0.077277	3.042772	1728.933110	0.024877
	0.211360	5.475601	4803.666690	0.000134
	0.111117	4.489795	4017.666630	22.904655
	0.255390	27.455572	6852.433320	0.000000
	0.071943	4.413971	2703.299870	18.912458
	0.110546	4.122207	3949.933470	0.256676
->	0.077030	2.912704	2550.899960	0.036215
	0.098790	2.944907	2904.633480	0.228167
	0.184869	14.841856	6478.733370	0.035016
->	0.082478	2.908699	2580.466610	0.033012
	0.081929	4.309336	2742.333370	0.018051
->	0.071987	2.768710	1715.233150	0.026106
	0.311648	10.596071	6966.766660	0.126593
	0.147693	16.588406	7068.833330	0.201431
	0.315992	4.745694	6152.766650	0.032509
	0.196606	13.813396	5723.099980	0.000000
	0.177056	7.255970	5922.333300	0.255579

->	0.080936	3.928435	3105.466610	0.031789
	0.146409	5.760113	4336.966550	0.000000
	0.142958	17.728558	6225.500030	0.000602
	0.638324	4.352890	6217.133330	0.035465
	0.205741	4.038198	4620.499880	0.000667
	0.195879	20.053480	6727.866670	0.156030
	0.114469	14.020228	4987.333370	0.035560
	0.094764	4.847567	3728.933410	0.000194
	0.358695	6.190195	6705.700000	0.036600
	0.196581	18.928171	7079.400000	0.171921
->	0.078153	3.200107	2016.666870	0.028479
	0.196745	5.675281	4943.566740	0.000056
	0.091926	4.660229	3193.466800	0.021769
->	0.095985	3.058718	2663.399960	0.034627
	0.127509	4.392328	4167.399900	0.000171
	0.158551	16.065483	6170.533370	0.033674
	0.097213	5.178959	3771.566770	0.000056
->	0.087344	3.846602	3091.199950	0.031896
	0.089510	3.761667	3498.766780	24.848162
->	0.073531	3.186256	1903.533330	0.028963
->	0.070090	2.831940	2002.366940	0.032866
	0.101542	4.937004	4326.166690	19.308685
	0.201965	9.601380	6113.866650	0.149407
->	0.078926	2.943133	2083.966670	0.028086
	0.186826	7.822977	5895.399930	0.036648
	0.270639	6.964880	6489.433360	0.037301
	0.086304	3.544233	3298.666690	18.475163
	0.104754	5.008738	3878.933410	0.031836
	0.170998	9.866519	5392.766720	0.000000
	0.137227	13.468022	5603.300020	0.000000
	0.222392	21.761541	7050.933330	0.030785
	0.465270	4.433714	6392.799990	0.033567
	0.097381	6.441845	3935.066530	0.000185
	0.086193	3.946472	3443.900150	29.392681
	0.113968	4.536852	4377.333370	0.000208
	0.088302	3.487366	3382.233280	18.749578
	0.358365	6.108839	6108.366700	0.000153
	0.074838	3.242904	2694.433290	25.975269
	0.104744	3.572748	3926.000060	29.720606
	0.114475	4.364361	3998.299870	0.000000
	0.166930	4.772071	4450.966800	0.000000
->	0.082112	3.114929	2672.066650	0.030296
	0.187665	4.741127	4485.466610	0.003106
	0.141757	15.062550	5698.466640	0.034995
->	0.078050	2.443440	2072.999880	0.028766
	0.237255	5.652805	6086.600040	0.044366
->	0.102202	3.736198	3911.533200	0.033176
	0.176405	5.211039	5203.000030	0.038750
	0.164803	4.512019	5379.666600	0.143725
	0.090881	4.912820	3740.599980	19.811543
	0.187813	11.046568	5590.899960	0.022363
	0.286073	3.802347	5349.766690	0.033456
	0.081119	3.914405	3180.299990	0.859486
	0.224669	27.682968	7146.933330	0.000185
	0.158823	6.865313	5224.900050	0.002060
->	0.071200	3.392723	1770.200200	0.027435
	0.158883	4.780229	4350.866700	0.000000
	0.127708	7.235388	4282.699890	0.000037
	0.150468	14.319886	5669.733280	0.037377
	0.143251	7.540841	4976.133270	0.144537
	0.428082	5.226823	6892.233330	0.032025
	0.258514	5.458655	6248.033370	0.040032
	0.173175	18.330124	6201.999970	0.000032
->	0.110273	3.927333	3393.566590	0.024722
->	0.084396	3.496424	2872.366640	0.029157
	0.290353	4.974261	5192.466740	0.003190

	0.195135	4.193498	4566.700130	0.037384
	0.301524	3.991582	5785.766600	0.136755
	0.195724	5.356804	5423.899990	0.034248
	0.262158	3.327318	4932.433320	0.033463
	0.465672	9.404505	7116.500000	0.038845
	0.409282	5.672105	6237.200010	0.000000
	0.770776	6.192739	6581.666680	0.031431
	0.082492	2.985482	2854.333190	0.239176
	0.119943	9.015676	4679.166720	0.000000
	0.090574	3.335271	3162.266540	0.223141
	0.324166	4.572963	5662.033390	0.034826
	0.382324	5.215993	5885.433350	0.000042
	0.101867	10.958806	4556.700130	0.036292
	0.291532	5.778160	6313.233340	0.002375
	0.366743	19.637897	7138.033330	0.154313
	0.398475	7.230349	6643.366660	0.000056
->	0.072885	3.067424	2213.566590	0.029475
	0.079348	4.234262	3040.366520	21.605257
	0.238101	5.513823	5647.299960	0.000361
	0.220367	3.803610	4429.266660	0.032697
->	0.079607	3.724039	1959.066770	0.024595
	0.198883	4.648989	5215.933380	0.035884
	0.331840	4.443915	6326.766660	0.030757
->	0.085571	3.164672	2267.333370	0.026667
	0.291318	9.420722	6437.266690	0.000000
	0.336451	6.626372	6950.533330	0.114375
->	0.110186	3.440856	3204.733280	0.032019
	0.440469	4.433465	6090.866700	0.035924
	0.115707	6.640939	4435.799870	0.000176
	0.272630	12.630587	6468.466640	0.000106
	0.158906	13.452278	5613.699950	0.034067
	0.120514	9.957024	4774.966740	0.000000
	0.284243	4.666306	4495.066530	0.000102
	0.181776	5.508034	4957.133330	0.000528
	0.117499	5.525406	4031.633300	0.000171
	0.096241	5.333781	3870.799870	0.026625
	0.117531	3.295798	3759.299930	0.139815
	0.075823	5.386506	2988.433230	0.236792
	0.111167	8.172285	4383.633420	17.675913
	0.344622	9.127618	6787.466660	0.035009
->	0.073664	2.915197	1949.233400	0.026715
->	0.082297	2.944597	2041.833500	0.023106
	0.485080	3.643364	5580.233310	0.034558
->	0.074703	3.619691	2621.199950	0.043053
->	0.081441	3.375690	2593.066710	0.027067
	0.323946	19.314249	7003.733330	0.000319
	0.170162	12.106651	5999.599990	0.035796
	0.225624	12.061184	6407.533340	0.045493
	0.201896	5.677804	5238.899990	0.000000
	0.217403	5.672579	5028.699950	0.000000
	0.210662	20.473927	7088.000000	0.034981
	0.077935	4.673529	2915.433350	0.000000
	0.098674	5.356060	3857.999880	0.000000
	0.297789	6.249142	5826.033330	0.000000
	0.100963	5.682649	3968.133240	0.025463
->	0.106735	3.587229	3023.800050	0.029266
	0.262419	20.933962	6879.000020	0.000190
	0.188304	12.375067	6304.599990	0.035648
	0.260648	6.442537	6420.566640	0.000000
	0.186007	15.968978	5931.466670	0.000000
	0.264761	3.966299	5438.999940	0.033544
	0.091814	5.984135	3680.499880	0.024813
->	0.133256	3.318030	3433.433230	0.027340
	0.135681	9.842681	5325.800020	0.034787
	0.225354	27.494074	7199.500000	0.000000
	0.367423	4.816131	5855.233310	0.001486

->	0.079651	2.797768	2131.433410	0.026720
	0.201126	3.837906	3981.833190	0.029567
	0.215881	13.270284	6615.000000	0.031410
	0.129218	9.514103	5294.100040	0.159315
->	0.085395	3.925315	2938.966670	0.027644
->	0.125671	2.818734	2975.199890	0.026894
	0.090639	3.363204	3621.666560	25.868908
	0.089256	4.294303	3556.633300	0.167132
	0.240207	5.719955	5367.400050	0.000000
	0.218171	20.280954	6781.933330	0.032275
	0.141419	11.660628	5337.200010	0.037794
	0.122607	4.847897	4507.166750	0.000079
	0.095440	6.277286	4057.966610	0.026227
	0.075595	3.479733	2960.133360	25.614222
->	0.081245	3.262640	1832.233280	0.005053
	0.078167	4.282127	3056.533200	22.508505
	0.253292	22.492622	7129.366660	0.051306
	0.137618	18.406523	7156.433330	0.183720
	0.123981	4.068509	3919.700010	26.864450
	0.206542	3.755182	2607.166750	0.006523
	0.356388	3.538597	5065.166630	0.032891
->	0.084526	2.886003	2148.900150	0.025484
	0.088188	4.444841	3665.066530	0.157988
->	0.073057	3.288718	2228.733220	0.033273
	0.394616	6.530215	6608.166660	0.000000
	0.073095	3.483655	2770.366520	19.266588
	0.169290	21.194857	6296.933360	0.049181
	0.095868	4.432713	3340.766600	0.000153
	0.120701	6.217556	4012.133480	0.000000
->	0.145178	3.832266	4395.666810	0.040410
	0.078910	3.216644	3059.166560	21.718834
	0.286981	6.356273	6041.100010	0.033113
	0.196719	17.830254	6921.966670	0.051123
	0.160831	11.929644	5141.966710	0.000190
	0.146811	8.152633	5279.100040	0.024593
->	0.086107	2.890382	2802.500000	0.035380
	0.117559	7.339276	4583.166810	0.003546
	0.740505	4.023455	5991.699980	0.033993
	0.084359	4.789759	3013.566590	0.000106
	0.222427	4.969787	4965.000000	0.001333
	0.234361	10.728419	6460.633320	0.031366
	0.488537	6.353385	6645.033340	0.027204
->	0.093113	3.770827	3405.266720	0.032127
	0.298569	4.607231	5154.166720	0.000000
->	0.078560	2.886709	2097.200010	0.029153
->	0.081105	3.882784	2809.133300	0.032500
	0.093412	7.929063	3961.000060	0.000000
	0.156193	6.595695	4996.233370	0.000000
	0.144855	4.190195	4038.966670	19.995718
	0.116475	3.439633	3287.133480	0.839380
	0.258534	5.121312	4792.700040	0.000000
	0.141577	13.598656	5837.599950	0.053319
	0.296348	9.685772	6596.800000	0.000000
	0.166510	19.526556	6875.533330	0.174130
	0.103885	5.191892	3745.400090	0.000000
	0.251358	27.991718	7200.000000	0.000208
	0.076179	3.779242	2789.866640	0.257130
	0.252171	3.590072	4471.566770	0.032106
	0.247645	6.222779	6262.666700	0.039178
	0.076119	4.000607	2776.166690	25.152298
->	0.074229	3.541429	1780.866700	0.023715
->	0.083265	3.431887	2103.066710	0.023303
	0.174210	4.473197	5020.633390	0.039479
	0.343766	8.439275	6901.866660	0.037211
	0.079046	4.554770	3419.566650	20.300913
	0.168964	12.751604	6196.866680	0.034988

	0.087457	4.078993	3172.799990	0.032178
	0.289402	10.506582	6427.433320	0.033049
	0.180543	17.765306	6541.600040	0.053403
	0.163716	4.757383	4402.999880	0.000000
	0.101057	4.874676	4078.566590	23.783274
	0.160151	9.598970	5402.666630	0.003505
	0.154105	15.952764	6400.166700	0.188132
	0.106563	12.795372	4297.166750	0.000185
	0.106141	8.382743	4515.233460	0.024231
	0.280538	5.974291	5688.633270	0.000153
	0.176318	21.460972	6157.066650	0.000000
	0.123889	7.874401	4785.200040	0.028447
	0.082728	3.616150	3280.233460	25.311323
	0.117655	7.891411	4764.633330	0.026810
	0.148659	7.168437	4693.233340	0.000000
->	0.070829	3.141914	2135.833440	0.038711
	0.107685	4.623549	3004.566650	0.000000
	0.078633	4.562937	2925.466610	18.809538
	0.089339	4.250891	3457.266540	17.772486
->	0.077866	3.178069	1987.566530	0.027424
->	0.085865	3.315548	2877.300110	0.029764
	0.111167	4.796078	4122.833250	0.276403
	0.085507	3.729663	3267.533260	0.240738
	0.166452	10.322081	5841.399990	0.130051
	0.224505	4.714035	5251.666720	0.031586
	0.252665	5.150788	5317.733310	0.000338
	0.348887	20.779663	7127.333330	0.166356
	0.186193	7.866537	5796.799930	0.037377
->	0.081215	2.815497	2048.400270	0.023766
	0.407495	7.072224	6813.800010	0.037546
->	0.078575	2.923535	2504.133300	0.032630
	0.134487	4.318389	3896.333310	0.031736
	0.228326	3.935208	5091.699980	0.033949
	0.176842	17.956362	6590.400010	0.000000
	0.124624	7.471703	4459.366760	0.000000
	0.153568	12.132035	5167.799990	0.000185
	0.326882	10.681141	7060.433330	0.000000
	0.238569	19.533554	6958.400000	0.031838
	0.135398	4.238501	3988.500060	0.031076
	0.090522	4.117125	3034.333190	0.028942
	0.146278	9.535438	5266.333310	0.035086
	0.199761	5.050543	4579.166560	0.000097
	0.108683	10.224491	4071.033330	0.000185
->	0.088256	2.880984	2202.000120	0.027255
->	0.082938	3.712420	3051.033330	0.030472
	0.089213	6.438528	3363.599850	0.000144
->	0.077876	3.043437	1975.033570	0.025602
	0.081567	3.646860	2996.199950	0.255345
->	0.074113	3.273577	1742.100220	0.025498
	0.140298	7.071510	4672.899930	0.000000
->	0.074162	2.872223	2428.299870	0.032375
->	0.071549	3.792227	1971.333620	0.029734
	0.300442	10.441777	6919.699990	0.000000
	0.203094	26.012627	7053.500000	0.149470
	0.110791	9.562461	4307.733460	0.000000
->	0.078071	3.113672	2505.966800	0.033245
->	0.084100	3.939233	2459.166560	0.014861
	0.141098	17.945171	6259.833300	0.150569
	0.077815	4.168735	2435.633240	0.005965
	0.122207	8.112834	5010.700070	0.163150
	0.253865	15.361819	6451.900020	0.000000
->	0.080509	3.930657	2450.266720	0.021685
	0.141646	11.445749	5245.933380	0.033340
	0.098159	4.811917	3660.833440	0.000208
->	0.097919	3.577524	2255.266720	0.016810
	0.098282	5.232728	3715.733340	0.000000

	0.283803	11.720780	6806.399990	0.000208
	0.365166	4.582450	6039.333340	0.026787
	0.300036	5.042267	5438.633270	0.029648
->	0.090901	2.444361	1992.500000	0.026461
	0.215848	5.788708	5811.600040	0.039852
	0.328132	3.924232	5285.233310	0.029097
	0.203909	5.579766	5005.366670	0.000000
	0.358941	7.199223	6546.566700	0.000000
	0.260190	6.420764	6448.466640	0.040729
	0.199650	6.408276	5263.000030	0.000264
	0.486367	3.483807	5304.499970	0.033359
	0.165076	11.049400	5802.466740	0.145375
	0.330980	4.696335	6556.033330	0.031949
	0.301347	5.106495	5817.133330	0.032424
->	0.086239	2.848985	2443.566590	0.028905
	0.089490	5.510273	3494.733280	0.000000
	0.139348	14.893899	6166.933360	0.045938
	0.110667	6.872192	4290.100100	0.000185
	0.245006	14.382654	6561.199990	0.001537
	0.171415	21.377731	6572.033350	0.159303
	0.082113	5.685102	3412.666630	0.028544
	0.085021	4.820104	3081.033330	0.031352
->	0.085856	3.805350	3076.900020	0.030748
	0.234657	6.001910	5416.000060	0.000139
->	0.080080	3.067794	2061.266480	0.025738
->	0.089120	3.996341	2601.499940	0.016343
	0.096225	4.615229	3345.366520	0.000208
	0.253786	4.831155	5852.733310	0.040236
->	0.082157	2.684271	2357.833250	0.032322
	0.114323	4.377584	3423.333440	0.000000
	0.207296	6.897863	5308.999940	0.000000
	0.182080	8.817625	5369.400020	0.026403
	0.147342	5.377743	4318.466800	0.000000
->	0.104140	3.094291	2784.166560	0.033954
	0.088734	3.260580	2931.366580	0.219310
	0.170156	5.188488	5164.299930	0.002616
	0.082903	4.921038	3451.866760	0.160350
	0.181469	5.884099	5145.233310	0.000185
	0.090850	6.785554	4065.299990	0.025620
	0.412521	8.239590	6907.766670	0.036905
	0.214607	5.182351	5410.500030	0.000056
	0.116987	8.260062	4123.633420	0.000157
	0.075358	4.370192	2233.299870	0.015322
->	0.071076	2.878823	1741.766970	0.026678
->	0.070964	2.672000	1551.266480	0.025725
	0.169576	11.516129	6569.800000	0.132481
	0.090409	5.050671	3273.033450	0.000130
->	0.076553	2.987666	2058.833620	0.029812
	0.292316	4.012297	5343.999940	0.031262
	0.089925	6.825408	3766.866760	0.026685
	0.252876	4.650846	4467.000120	0.000361
->	0.073500	3.037425	2257.433470	0.031019
->	0.085264	2.967070	2206.433410	0.021368
	0.109067	7.319562	4312.566530	0.026447