AN ABSTRACT OF THE THESIS OF

Adam R. Bell for the degree of Master of Science in Mechanical Engineering presented on November 19, 2010.

Title: Learning-Based Control and Coordination of Autonomous UAVs

Abstract approved:

Kagan Tumer

Uninhabited aerial vehicles, also called UAVs are currently controller by a combination of a human pilot at a remote location, and autopilot systems similar to those found on commercial aircraft. As UAVs transition from remote piloting to fully autonomous operation, control laws must be developed for the tasks to be performed. Flight control and navigation are low-level tasks that must be performed by the UAV in order to complete more useful missions. In the domain of persistent aerial surveillance, in which UAVs are responsible for locating and continually observing points of interest (POIs) in the environment, such a mission can be accomplished much more efficiently by groups of cooperating UAVs.

To develop the controller for a UAV, a discrete-time, physics-based simulator was developed in which an initially random neural network controller

could be evolved over successive generations to produce the desired output. Because of the inherent complexity of navigating and maintaining stable flight, a novel state space utilizing an approximation of the flight path length between the aircraft and its navigational waypoint is developed and implemented. In choosing the controller output as the net thrust of the aircraft from all control surfaces and impellers, a controller suitable for a wide range of UAV types is reached. To develop a controller for each aircraft to cooperate in the persistent aerial surveillance domain, a behavior-based simulator was developed. Using this simulator, constraints on the flight dynamics are approximated to speed computation. Each UAV agent trains a neural network controller through successive episodes using sensory data about other aircraft and POIs.

Testing of each controller was done by simulating in increasingly dynamic environments. The flight controller is shown to be able to successfully maintain heading and altitude and to make turns to ultimately reach a waypoint. The surveillance coordination controller is shown to coordinate UAVs well for both static and mobile POIs, and to scale well from systems of 3 agents to systems of 30 agents. Scaling of the controller to more agents is particularly effective when using a difference reward calculation in training the controllers.

Learning-Based Control and Coordination of Autonomous UAVs

by
Adam R. Bell

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented November 19, 2010
Commencement June 2011

Master of Science thesis of Adam R. Bell presented on November 19, 2010.

APPROVED:

Major Professor, representing Mechanical Engineering

Head of the School of Mechanical, Industrial, and Manufacturing Engineering

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Adam R. Bell, Author

ACKNOWLEDGMENTS

I would like to thank my wife Lorcy for her patience and support. Without her encouragement I would likely have given up on myself long ago. I would also like to thank Dr. Kagan Tumer and Dr. Matthew Knudson for their support and guidance throughout this project, without which I would surely have been totally lost within mere weeks of entering graduate school.

For his patience while giving me a crash course in LaTeX, his interest in my research and its fit within the broader realm of control systems, and for being a friend, I also want to thank Cody Ray, and by way of Cody, Ben Dickinson for the use of his custom style files in building my thesis.

The rest of the AADI research group have inspired me to keep going and stay focused on my research, while also giving me the opportunity through their questions to realize just how much I have actually come to understand about learning control policies.

Table of Contents

List of Figures

1 Introduction

Uninhabited aerial vehicles (or unmanned aerial vehicles), also called UAVs, are becoming increasingly common in civilian and military sectors. Currently, they are piloted remotely by a human operator at some distance from the UAV. This remote piloting of the UAV means that constant contact is needed between the operator and the UAV unless some form of autopilot mechanism is used for control for all or part of a mission. Such constant contact has significant drawbacks, including a large energy expenditure by the UAV in transmitting sensor and image data, and perhaps more importantly, there exists a delay between the time an operator sends a command to the UAV and the time that the command is executed.

Aircraft in general are useful largely for their speed, broad viewing area, and ability to bypass rough terrain, making them useful for tasks such as topographical mapping, monitoring crops, detection of forest fires, and search and rescue. Many of these tasks are very similar and can be abstracted into a more general mission of persistent aerial surveillance, in which UAVs will continuously observe some points of interest in the environment which could model lost hikers, suspects fleeing from police custody, or even wildlife. This is a domain in which having multiple UAVs observing the same area can greatly improve the mission performance either by speeding up a search, or allowing greater coverage of an area to more closely observe more POIs. As the density of the UAVs in a search space increases, it is easy to see that coordination between them becomes necessary so that they are not getting in each others way and they are effectively using their numbers to observe many POIs.

To coordinate the UAVs, we want to develop a controller that will distribute the UAVs in the environment in such a way that they make good use of their own numbers in observing POIs. In performing this autonomous coordination of UAVs,

some other controller is still necessary for the lower level tasks of flying and navigating the individual UAVs.

Previous research has developed control methodologies for controlling the flight and navigation of a single aircraft using a variety of different means. Flight control can be achieved through a feedback control loop which can vary from a Proportional Integral Derivative (PID) controller to more complex forms of control such as Linear Quadratic Regular (LQR), and optimal control methods like receding horizon control [22, 17]. These controllers are derived from the equations of motion governing a system, meaning that any deviation from the expected equations of motion may produce undesired behavior if not accounted for while developing the equations. Traditional control methodologies such as PID typically need to be highly tuned for many applications including flight control, which means that a PID for one type of UAV may not work well for a UAV with different control surfaces. Also, a PID controller must be carefully designed to take into account any sort of disturbance to the system model, but more complex controllers like LQR can build in greater tolerance, meaning that a UAV may be more robust unexpected forces such as wind gusts or ground effects. However, to achieve precise control of a UAV a detailed model of the system including all control surfaces and impellers must be developed, so any shortcomings in the model will always be present in the controller.

Learning-based control methods such as an artificial neural network have been shown to be able to achieve level, stable flight for conditions under which traditional control methods struggle or even fail [44, 42, 43]. This is due to limitations that are essentially built-in to traditional control methods by approximations that must be made in deriving these so that an analytic solution can be found. A common example of such an approximation is the linearization of nonlinear terms in equations. This frequently occurs when developing a state space form of a system to put into

matrices, which must be linear for these methods to work. Linearizing specific system parameters means that the control laws derived will only be valid for a certain range of values that the system parameter can hold. Engineers, mathematicians, and physicists are all familiar with the Small Angle approximation, where the $sin(\theta)$ or $cos(\theta)$ can be approximated as θ for a small range of values that θ might take.

A learning-based control method requires no specific model of the system. Instead, from an initially arbitrary function, a controller that maps input parameters to outputs is morphed to work well for the system according to some signal that provides feedback about its performance. This signal is often in the form of a function, designated as the system utility function or objective function, that quantifies the behavior of the system and can be minimized or maximized to achieve the desired system performance. The set of all possible values that the inputs can take is referred to as the *state space* of the system [41]. Since the values of the table entries or function variables are not derived, approximations like the Small Angle approximation are not ingrained in the control laws, and the resulting fully-trained controller will often handily deal with highly nonlinear system dynamics that traditional controllers fail in [44].

Such feedback control works well under ideal conditions as such conditions are accurately described by the model produced from the equations of motion. UAVs of the scale typically operated today are large enough that their sheer size and mass make them robust to moderate disturbances such as high frequency wind gusts and turbulence [42]. However, while traditional feedback control methods may work well for larger UAVs, MAVs suffer from a dramatic reduction in aerodynamic stability due to their much smaller size. This means that the equations of motion from which traditional control schema are derived are often too simplified to effectively control any MAV that is subjected to disturbances such as wind or other atmospheric effects

such as precipitation [5]. In order to create a more robust controller, such disturbances must be accounted for, either explicitly or implicitly, within the control system.

One of the most prevalent uses for UAVs is persistent area surveillance, where monitoring of an area is performed by one or more aircraft [29]. It is not difficult to imagine a single aircraft thoroughly searching tens of thousands of acres of rugged terrain for lost hikers taking several days. This search time can be improved by increasing the number of aircraft involved in the search, so that several different aircraft could be searching in parallel allowing faster, more efficient searching. The question then becomes how best to distribute the aircraft so that an efficient search can be conducted.

This coordination can be achieved either through a centralized controller or by a distributed control system. A centralized controller would be an algorithm or human that would use the positions and headings of each aircraft and use that to decide how best to organize the aircraft. A distributed means of controlling the system could be constructed in many different ways, depending on how an agent is defined. For example, an agent could be defined as a fixed navigational waypoint, and its action could be to determine which agent or agents are assigned to it. The UAV would then simply fly to the designated waypoint. Alternately an agent could be defined as an aircraft, as is done for this research, and its actions would be to select a waypoint, at which point it would, again, fly to that designated waypoint.

Similar coordination of UAVs for the persistent aerial surveillance task has been achieved in several different ways. One such method is to formulate the problem in such a way that a global optimum solution can be found using mixed-integer linear programming (MILP) as in [15, 37, 38, 6]. In order to formulate the problem in this way, Similarly, an alternate formulation divides the search space into an array of smaller regions, so that an optimum solution can be found by minimizing the elapsed

time that each region is visited by a UAV [29]. These approaches to coordination of UAVs are good because they find the optimal solution, but a number of assumptions must be made to formulate the problem in such a way that a solution can be found with this method at all. The assumptions made include exact knowledge by the controller of every UAV in the system, the UAVs always do exactly what they are told regardless of circumstances, and that communication costs are trivial. This last point about the communication costs is perhaps the most important, as communication becomes very costly as the size of autonomous aircraft continues to shrink.

Many different types of coordination problems between UAVs have been studied previously, but the most commonly studied among these are path planning of multiple UAVs [14, 51, 16], formation flight [34, 33, 46, 49], target tracking [17], and concurrent rendezvous [8, 26]. The control systems designed for many of these are distributed controllers, meaning that each UAV uses the information it can gather to make its own decisions, creating reliable and resilient controllers [35].

To create these control systems, most of the focus was on either path planning of an individual UAV or on the creation of stable formations of UAVs. In the case of path planning, the overall approach is best typified by [8], in which a UAV must choose the best path between its current position and a target position. This is done by choosing a series of short path segments based on their relative value to the UAV, decided by a function relating the speed of taking that path and the danger associated with it. The authors then offer an extension of this work that can apply to multiple UAVs by including in the path value calculations the proximity of other UAVs. An approximation is inherent here because of the coupling of the spatial and temporal relations of the UAVs in the environment, so that in choosing a path to follow a UAV must forecast the future positions of other UAVs.

Formations of UAVs are useful because of their ability to allow UAVs to fly in close

proximity and avoid collisions that would be catastrophic to the aircraft. In [34, 33] these formations are assumed to be planar in order to simplify the mathematics of the graph theory and potential functions between UAVs, limiting somewhat the applicability of this approach. However in [46], formations capable of existing in three dimensions are shown to be able to be controlled. While formation flight of UAVs has benefits, it does not characterize a useful mission of the UAVs of itself, and this is left for other research.

The ultimate goal of creating fully autonomous UAVs is to allow a single human user to be able to easily control large groups of UAVs using very high level tactical commands. In this research we will first show that an individual UAV can autonomously perform low level tasks including stable flight and navigation to waypoints. We then build on this to show that using a distributed system we can give a group of UAVs a very general goal and they will be able to use the limited information available to them to carry out their mission.

One of the most important features of a distributed system is its fault-tolerance, or robustness to failure. In order for the control of a group of aircraft to be as robust to disturbances as possible, a decentralized control scheme should be used. Agents that are controlled reactively can cope with a change in the number of agents without having to learn a new control policy, where a centralized controller must generally reprocess.

The research presented in chapters 3 and 4 will show how a single artificial neural network controller can be evolved that can both achieve stable flight and navigate to a waypoint. In order to do this, a novel state space that models UAV sensory data is developed that is then used as input to the neural network controller. From these inputs, the neural network determines action for the UAV to take in the form of an applied force vector that can be thought of as the combination of all of the forces

applied to the aircraft by its control surfaces, whether these forces arise from elevators and ailerons for a conventional fixed-wing airframe, adjusting main rotor pitch and tail rotor speed for a single rotor helicopter, adjusting individual rotor speeds for a quad-rotor, or from the use of jets of air [36] as from a flapless UAV like the Demon, developed at Cranfield University.

To create the controller, a simulated environment was necessary to capture the essential dynamics of the aircraft operating in the environment, so that at each time step in simulation, the acceleration, velocity, and position of the aircraft are updated according to realistic physics. Starting from a population of neural networks, each with randomly assigned weights, the population is evolved over successive episodes using the neuroevolution algorithm. After a neural network is used for an episode, its performance is evaluated according to an objective function that measures how well the neural network was able to control the aircraft both in terms of improving its trajectory and position with respect to a waypoint. For instance, if a waypoint is directly behind the aircraft initially, it will be critical to reward a network that is able to steer the aircraft towards the waypoint, as the physics of the system dictate that at least for a certain amount of time the aircraft must be moving away from the waypoint. These two disparate types of information will be put together using an approximation of the aircraft path length to reach the waypoint from its current position and using its current velocity.

In chapters 5 and 6, a higher level controller will be developed to coordinate agents by assigning waypoints to each agent based on sensory information about its environment. Using a multiagent systems approach, agents will coevolve their neural network controllers. Agents need not communicate directly, but will use sensory data to make reactive decisions about where to go in a simulated environment by treating other agents as though they were part of the environment.

The objective of the team of UAVs here is to perform persistent surveillance of an area by tracking either static or dynamic points of interest in the environment, similarly to [19] but with significantly different dynamics than the terrestrial rovers studied in that research. Such points of interest could represent such things as stranded hikers, enemy troop emplacements, or police suspects fleeing custody. Here a different sensory setup must be used to encapsulate information not only about the points of interest (POIs), but about other aircraft. Sensors give a rough estimate of where other agents and POIs are relative to the agent, which will again be used as inputs to a neural network that maps these inputs to outputs consisting of the axial location of a waypoint for the agent to approach. The action selection for this was specifically chosen so that the controller for an individual agent as described above (and in more detail in Chapter 5) can be used to navigate the agent to the waypoint described by this controller output.

In the following chapters, we will provide more specific examples of what has been stated above. Chapter 2 will provide a solid background for this research, including different control methodologies for flight control and for coordination of autonomous agents. Chapter 3 presents the development of a neural network controller that can achieve stable, level flight of an aircraft while navigating it between waypoints. Chapter 4 presents the results of applying this controller in a simulated environment will be presented. Chapter 5 will show how for each agent in the system, a controller can be evolved that is capable of coordinating the agent within a group of autonomous aircraft, and experimental results from this in Chapter 6. Finally, Chapter 7 will discuss the implications for our work and propose future directions for the research.

2 Background

2.1 Control of an Autonomous Aerial Vehicle

In the ongoing transition from remote operation of UAVs to fully autonomous control, controllers must use sensory input collected from the aircraft to decide on an action. This can be done in any of several ways. The most widely applied form of control is traditional feedback control theory, which uses equations derived from the system dynamics along with the sensory information to determine the action to take for given sensor values. This need for a system model can limit the generalizability of the solution obtained because of the mathematical approximations necessary to arrive at an analytical solution [39]. A model of a the system is used in modern control theory as well, including the various forms of optimal control that have been developed over the past several decades [27].

Learning methods such as reinforcement learning and neuroevolution are constructed such that over time, an initially random mapping of sensory inputs to actions is improved using a reward signal that measures the performance against the desired behavior [41]. Reinforcement learning methods achieve this through the use of a table over the possible state space values, and each entry in the table is updated as the state space is gradually explored. Neuroevolutionary methods perform a similar update over time, but rather than a table, a function approximator in the form of an artificial neural network is used and the weights between network nodes are changed over time. As these learning algorithms do not use an explicit model of the system, the behavior that arises as they explore the state space can work well under a variety of conditions, often including conditions under which they have not been trained [41].

2.1.1 Model Predictive Control

As suggested by the name of this class of control methods, to develop a controller a model of the system to be controlled is developed. This model takes the form of mathematical equations describing the dynamics of the system. By defining which parameters are to be controlled, a feedback mechanism is used to set the free parameters in the equations to achieve the desired behavior according to the mathematical model used. Thus, from the values of the free parameters, we can predict the output variables using the mathematical model.

In [21], control coefficients for parameters including angle of attack, roll rate, pitch rate, and others were determined for a specific UAV (in this micro air vehicle or MAV) geometry and using the characteristics of the airfoil. In order to control the aircraft, closed-loop control laws were derived using root-locus techniques. In simulation, it was shown that this worked well and could be generalized slightly to include airframes with slightly different centers of gravity and geometries.

For complex systems with significant nonlinearities, approximations must be made to develop systems of linear equations so that a constant coefficient matrix can be formed. The matrix vector form describing the system state is a prerequisite for designing a controller. Constructing such a mathematical model requires intimate knowledge of the internal processes of any system in order to find the equations that describe these processes [27, 25].

2.1.2 Reinforcement Learning

In Ng et al. [28], a reinforcement learning algorithm was implemented and was able to achieve sustained inverted flight of a helicopter, not just in simulation, but for a

physical airframe. Helicopters are considered to be highly unstable due to their very non-linear dynamics and noisy parameters, but the complex task of performing certain maneuvers was accomplished by first developing a model of the system dynamics and using this to train their own *PEGASUS* reinforcement learning algorithm so that it did not have to learn every facet of the complex system behavior from scratch.

In a domain more similar to the control and navigation of an uninhabited aerial vehicle (UAV), reinforcement learning algorithms have been used for sensory-action pair mapping of an autonomous underwater vehicle [9]. In this paper, a reinforcement learning algorithm is used to develop a probabilistic control policy by changing the weights of a neural network. In a two-dimensional simulation, their underwater vehicle agent is able to successfully follow a target using the control policy developed.

2.1.3 Neural Network Control

Neural networks are often referred to as universal approximators [7]. Because of the way these networks are structured, they can be developed such that any nonlinear function can be approximated. The structure of an artificial neural network (NN) a layered structure, with input nodes on one end, output nodes on the other end, and some number of hidden layers in between each consisting of some number of nodes. This is easier to conceptualize in looking a general schematic such as in Figure 2.1.3.

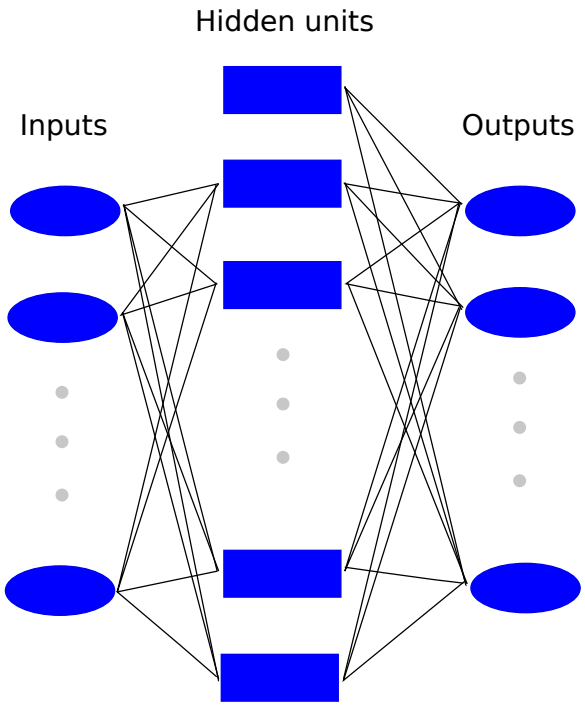


Figure 2.1: Schematic of the generic structure of an artificial neural network, in which input values are fed through the structure on the left and output on the right after some mathematic manipulations. In learning of the control policy, the weights connecting nodes are updated using a reward signal.

Each node is connected to the nodes in the following layer by a variable weight, which is initially a random value. Each node in the hidden layer has an activation function so that as the value of the combined nodes coming into it are summed together, the function determines whether or not a threshold has been reached for the node to be active. The most common activation function is a sigmoidal function as in Equation 2.1 due to its continuity and differentiability, but others can also be chosen such as the unit step function.

$$P(t) = \frac{1}{1 + e^{-t}} \tag{2.1}$$

In applying an NN, we may choose any number of hidden layers, each containing

any number of nodes. These choices effect the complexity of the network and thus its ability to capture highly nonlinear data distributions. The usefulness of applying the neural network control methodology has only begun to be addressed recently as its usefulness as a reactive control scheme is implemented and compared to other reactive schemes such as decision trees [40].

Using these networks as controllers and training them using an evolutionary approach has proven very successful in robotics domains closely related to UAVs such as terrestrial rovers [10, 12, 11]. While these rovers have very different dynamics than a system in flight, there are certain overlaps in the tasks they must perform in navigating such as obstacle avoidance. In relating sensory data to action selection, a wide range of options is presented in existing research. Inputs to the neural network can be the agent location [12, 11] and battery charge remaining.

Another similar domain using neuroevolution is control of a rocket [13] not using fins, but rather through changing the direction of the thrust. More relevant still is work done in using neural networks to stabilize flight in micro air vehicles (MAVs) [42, 43, 44]. For very small fixed-wing airframes, it was shown that a neural network controller was capable of stabilizing flight even under large disturbances such as strong wind gusts [42, 43]. For a very different type of aircraft, a quadrotor, it was demonstrated that a neural network controller combined with a highly nonlinear model of the system dynamics was capable of vastly outperforming a traditional feedback controller (PID, or proportional integral derivative controller) even in the face of a disturbance to the aircraft that flipped it completely upside down [44].

2.2 Coordinating Uninhabited Aerial Vehicles

Coordination of groups of aerial vehicles has much in common with the coordination of ground-based vehicles. The similarities include similar goals, restrictions on communication, and the dependence of action choices of an agent on the actions of other agents. The key distinguishing characteristics are the dynamics of motion, perhaps most exemplified by the inability of an agent to drop below a certain speed without catastrophic consequences. A direct consequence of this is a significant increase in the turn radius compared to the typical terrestrial robotics platform modeled by most research, in which an agent may come to a complete stop and turn in place if needed.

2.2.1 Traditional Control Theory

Mathematically derived feedback control laws have been derived that allow groups of UAVs to achieve stable, coordinated formation flight [34, 33, 46, 49]. Formation flight is potentially very useful for certain situations, as multiple UAVs are able to fly in close proximity without collisions, and can maintain their formation while performing maneuvers by adjusting the spacing between UAVs.

In [34, 33], this formation is achieved using structural potential functions that act to repel UAVs from one another, together with a formation graph that serves as a map of the coupled interactions of UAVs in the formation with one another. In setting up the control system in this way, each UAV can change its relative velocity to those around it using only information about its nearest neighbors, thus adjusting its position in the formation. In order to ensure stability of the solution, it is assumed that the formation of UAVs exists only in a single plane, so any three-dimensional formation is not allowed.

Similar work was done by [46] but was able to extend the usefulness by working for formations in three dimensions. This is done by defining subsystems within the formation that each use state feedback controller, so that individual UAVs only detect the vehicle directly ahead of them.

Such distributed control laws have been shown to be more reliable and robust to disturbances than centralized control of UAVs [35].

In [14, 51] and [16], the problem of path planning for multiple UAVs in hostile environments was considered. In both of these, UAVs must find an optimal path from their starting position in the environment to a target, while spending as little time in hostile territory as possible. In [16], this was done using an *a priori* probabilistic map of existing enemy troop locations and obstacles that may move over time. Both [14] and [16] perform this path planning for a single UAV and illustrate how the approach could be extended to multiple UAVs.

An alternative method for path planning uses $\mathcal{L}_1$ adaptive feedback for the coordinated path planning of multiple UAVs so that collision avoidance is built in to the controller [17].

Another coordination task addressed is the rendezvous of multiple UAVs at a predefined waypoint [8, 26]. In both of these, again what is essentially happening is the planning of a path for individual UAVs in order to minimize the amount of threat they are subjected to in the environment. To create the paths, Voronoi diagrams are used in conjunction with some algorithm (Dijkstra in [8]), essentially creating segmented paths such that the threat over each path segment is minimized.

Mathematically derived feedback control laws have been applied in countless applications. Such control laws are derived using a mathematical model of the system dynamics to be controlled as developed from physical principles such as the equations of motion of the bodies in the system. While the underlying assumptions on

which they are based hold true, such control laws are mathematically guaranteed to work [27, 25]. However, the soundness of such assumptions relies on the underlying dynamics of the system being modeled and in some problem domains these simplifying assumptions may hold only for narrow bounds of certain parameters [39], while some domains may present large difficulties in even finding an adequate model of the system from which to derive a control scheme.

Optimal control techniques such as receding horizon control have also been applied in this domain for assigning trajectories or other tasks to UAVs [22, 17]. These controllers must be based on physical constraints of the system, so a controller developed for fixed-wing airframe in this fashion would not be able to be applied to a different type of aircraft.

2.2.2 Numerical Optimization Techniques

In order to alleviate some of this problem, other control methods can be used. Optimization techniques such as mixed-integer linear programming (MILP) have been used for control of not just single aircraft , but also for control of teams of aircraft [37, 38, 6, 15]. These papers formulate the coordination problem as essentially having two parts: a task allocation component, and a trajectory planning component. With the goal of minimizing the total mission completion time of heterogeneous UAVs visiting specified waypoints, they were able to formulate the problem into a solvable form by simplifying the vehicle dynamics and putting these and the problem constraints into matrix form. MILP was then used to find the optimal solution in the form of setting waypoint and assigning UAVs to them.

Optimization methods, as with feedback control laws, can work well in some instances. The main drawback with the traditional approach to these methods is the

explosion of the state space. As more and more variables are considered in determining the overall state of the system, finding an optimized solution can sometimes mean searching through hundreds of billions of possible solutions , which can become computationally intractable [45, 50].

A related method develop in Nigam et al. [30, 29, 31] is actually tailored for the persistent aerial surveillance task. In these papers a method is developed and extended from one to two dimensions of maintaining temporally measured observations of the entire map. To do this, the map was divided up into cells, and for each of these the time since its last observation is recorded. In deciding which cell to visit next, or rather, what path to follow, an algorithm is developed that optimizes the path to minimize the average "age" of each cell, where the age is simply the amount of elapsed time since it was last visited.

2.2.3 Learning Methods

Machine learning algorithms offer a form of adaptive control possessing some properties that overcome the shortcomings of both feedback control laws and optimization techniques. Principally, the lack of a need for a system model means that although there may be no guarantee of the control policy working for all scenarios, no simplifying assumptions are present so the control policy will often work outside the range of input parameters for most feedback control laws [44], although demonstrating this is domain specific. While some machine learning algorithms fall prey to a state space explosion as optimization methods do, many algorithms such as neuro-evolution, can easily overcome this through some clever representation of the state information [32]. There is of course a trade-off between the representation of state space and the performance level that can be achieved. Generally, if we decrease computational cost,

we are also decreasing the performance that can be attained by the system.

Using a multiagent systems approach can lead to control systems for coordination that are robust to agent failures and very adaptable to dynamic environments [19, 18, 20]. This approach prescribes that each agent in a system, or each aircraft in a team of autonomous aerial vehicles, has its own control policy so that as the system changes, far less computation is required to reconfigure the system meaning that such reconfiguration can be done in real time. In [40], an evolutionary scheme was applied to develop a reactive controller for coordinating groups of micro air vehicles where all team members share a target that they must reach at which one will simulate a detonation and the target will be destroyed. The sensory information to these MAVs is provided by a central system that tells them at each time step the location of the target in the environment.

In [50], the task of persistent aerial surveillance was addressed for heterogeneous UAVs with simplified dynamics by having a centralized controller that learned to adjust the distribution of UAVs in the environment to observe static or dynamic features.

3 Navigation

The algorithm used for simulating the flight of a single autonomous uninhabited aerial vehicle (UAV) has a very simple main control loop for program execution. This algorithm is just an instance of the traditional robotic control loop of sense, process, execute. The general idea behind the state space representation used here is to quantify the value of multiple possible flight paths surrounding the current trajectory of the UAV. Such a representation helps to distinguish between small deviations in trajectory for the short time-scales used here.

Learning occurs via neuroevolution, so the training consists of a series of episodes in which the agent takes takes actions for each state it encounters and is then rewarded based on its overall performance through the episode as shown in Figure 2.

3.1 Agent Description

The characteristics of the aircraft chosen for the simulation were modeled after a generic fixed-wing MAV. The flight characteristics of the aircraft deemed most critical for simulating the flight dynamics are the maximum speed, stall speed, and cross-sectional area, and are summarized in Table 3.1.

	DIMENSIONS	
W	Width of the simulated world	2000 (m)
H	Height of the simulated world	2000 (m)
D	Depth of the simulated world	1000 (m)

Table 3.1: Table of the spatial dimensions of the simulated environment.

3.1.1 Sensors and State Space

The sensors used for the agent to determine its state consist of nine paths arrayed around the current trajectory ($\vec{v}$) of the agent at each time step as can be seen in Figure 3.1.1. Since each of the nine paths is relative to the current trajectory of the agent, the paths themselves are easily calculated by adding or subtracting a fixed angle of $\frac{\pi}{4}$ radians for the inclination and $\frac{\pi}{4}$ radians for the horizontal trajectory. For each of the nine potential paths, the approximate path length S_i is calculated using the current speed of the UAV. This sensory setup does not correspond to any particular type of sensor, but is a useful abstraction that approximates the function of actual sensors.

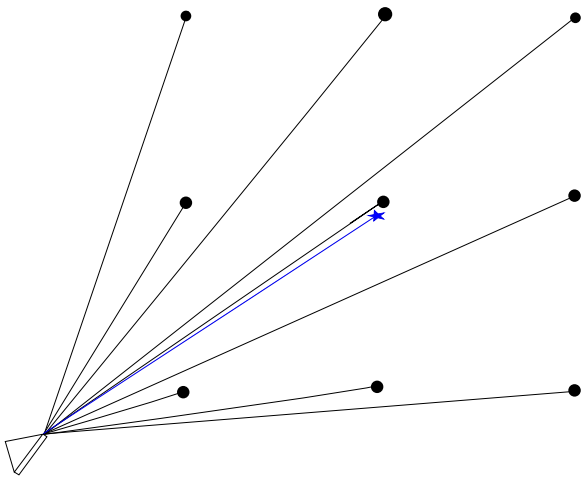
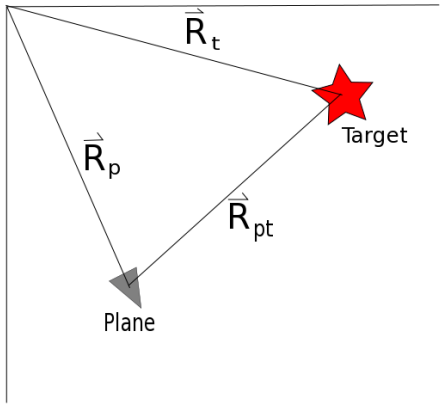


Figure 3.1: Sensory state setup for the aircraft agent consisting of nine potential paths surrounding the current trajectory of the agent. Each path has an associated value based on the approximate path length S_i.

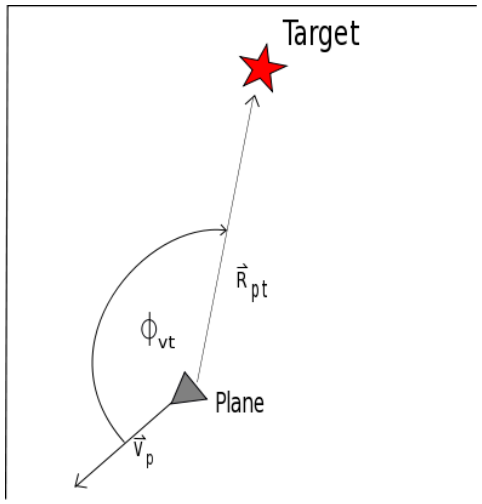
Each of these nine paths is assessed a numerical value based on an approximation of the length of the flight path the agent would have if it were to have that path as its trajectory. The basic concept here is actually quite intuitive: is that path shorter than the one we are taking? We can quantify this, but since we cannot know the actual length of the flight path ahead of time we must approximate it. If we designate the

actual flight path length down path i as S_i^{actual}, then the approximation of this is designated simply S_i, or the vector containing all nine sensor values as $\mathbf{S}$.

To calculate the approximate path length S_i, we will use a combination of the euclidean distance from the current position of the agent to the waypoint, and the arclength circumscribed by the vectors $\vec{v}$ and the relative position vector $\vec{R}_{pt}$ between the agent and the waypoint, shown in Figures 3.2(a) and 3.2(b).



(a) Schematic of the two-dimensional vectors of absolute position R_p and R_t of the plane and target respectively, and the relative position vector between them R_{pt}.



(b) Diagram showing ϕ, the angle between the vectors $\vec{v}_t$ and $\vec{R}_{pt}$.

Figure 3.2: Diagrams of the vectors used in path length calculations. Shown are the relative position R_{pt} and the angle between this and the velocity of the agent $\vec{v}$. Note that in simulation, these vectors are in 3D.

Using the angle ϕ between the vectors $\vec{v}_p$ and $\vec{R}_{pt}$, the arc length between them using the magnitude of the $\vec{v}_p$ corresponds to a term in the equation for S_i (Equation 3.1) that takes into account the speed at which the agent is moving relative to the waypoint. The other term in Equation 3.1 is simply the three dimensional euclidean distance between the agent and the waypoint. This gives us a single equation that takes into account the absolute distance between the agent and the waypoint and the

relative motion between the two, as shown in Equation 3.1.

$$S_i = |\vec{R}_{pt}| + \phi_{vt} * \vec{v}_p \tag{3.1}$$

In choosing this sensory setup the goal was to provide the learning agent with information regarding the value of its current trajectory and the neighborhood of actions that it could take from its current state.

3.1.2 Action Set

In order to keep the simulation as generalizable as possible, rather than using specific control surfaces and engine characteristics, all of the forces applied to the aircraft from control surfaces and engine thrust have been combined into a single three dimensional force vector. This allows the same control policy to apply equally well to many different types of aircraft, from fixed wing to quadrotor aircraft. In order to apply such a general control policy to a specific aircraft, some sort of function approximation can then be used to determine the appropriate positions of the control surfaces into the appropriate positions to generate the required conglomerate force vector. This vector of the sum of the applied forces, designated $\vec{T}$ for the total thrust of the aircraft, will be allowed to take on any vector value regardless of the current trajectory of the aircraft, normalized to a maximum thrust of the aircraft. For some types of airframes $\vec{T}$ may only be able to assume a subset of these values, but to keep this in the most general case such constraints will be neglected here.

3.2 Simulator

The simulator used to train the controller for the combined flight control and navigation task is a physics-based real-time simulator. This means that an episode in the simulator begins at some point in time with initial conditions including the position in cartesian coordinates of the waypoint and the agent, and the velocity vector ($\vec{v}_p$) of the agent. From this initial point, we iterate forward in time, updating the acceleration, velocity, and position of the agent at every time step by calculating the forces on it for its current velocity. The physics used here is simplified a great deal from most of the technically accurate flight simulators available because computation time will be important for this simulator as it will be used for tens of thousands of episodes at a time.

Besides the applied forces that arise from the control surfaces and engine of the agent ($\vec{T}$), the environmental forces assumed to be acting upon the aircraft are gravity, drag, and lift. In calculating these they we will simplify somewhat for the sake of reducing computational complexity, while retaining enough fidelity to provide relevant results from the simulation. The following equations show how these forces are calculated within the flight control simulator. The force of aerodynamic drag on the aircraft, F_D is a function of air density (μ), the speed of the agent, and the drag coefficient C_D which actually varies with speed but will be treated here as a constant in order to simplify the computation (Equation 3.2.

$$F_D = \frac{1}{2} * \Delta t * \mu * C_D * A * |\vec{v}_p|^2 \tag{3.2}$$

The force of lift is also a function of speed and air density, but the notable difference here is that it is dependent on the angle of attack of the aircraft, or the angle at which the aircraft is oriented, as in Equation 3.3.

$$F_L = \pi * \Delta t * \psi * \mu * |\vec{v}_p|^2 \tag{3.3}$$

Equation 3.4 is simply the force on the aircraft due to the constant acceleration of gravity.

$$G = \vec{a}_G * M * \Delta t \tag{3.4}$$

Each of Equations 3.2 to 3.4 are essentially operating in three dimensions, but in approximating them we assume that F_L is always acting in the positive z direction, while F_D always acts directly opposite the direction of the agent velocity $\vec{v}_p$. The numerous constants in these equations are tabulated in Table 3.2.

	PARAMETERS	
μ	Density of the air	1.297 (kg/m^3)
Δ_t	Time step	0.1 (s)
ψ	Pitch of the airfoil	
$\bar{a}_G$	Acceleration due to gravity	
$\bar{v}$	Agent velocity vector	
C_D	Coefficient of drag	1.8 $(unitless)$
M	Mass of the aircraft	100 (kg)
A	Cross-sectional area of the aircraft	0.6 (m)

Thus, at every time step, the acceleration of the agent can be calculated using Equations 3.2, 3.3, and 3.4, and the applied force vector $\vec{T}$ by summing the forces in each of the x, y, and z directions according to Newtonian mechanics. To obtain the velocity from this, we use a standard rectangular numerical integration scheme, chosen for its low computation cost, as in Equation 3.5. All that is stated in Equation 3.5 is that the new velocity is the sum of the old velocity and the product of the acceleration with the length of time that has passed. While this has a low order of accuracy, for

a sufficiently small time step it is numerically stable and used in nearly every flight simulator created.

$$\vec{v}_p^{t+1} = \vec{v}_p^{t} + \vec{a}_p^{t+1}\Delta t \tag{3.5}$$

In computing the new position, the same numerical integration technique is used again to go from the old position and new velocity to the new position.

3.3 Algorithms

The three algorithms presented here are all interrelated in the development of a flight and navigation controller. To establish statistical significance, each complete simulation must be run for a certain number of repetitions, generally accepted in the computer science community to be at least twenty to thirty runs. Each run consists of anywhere from several thousand to over ten thousand episodes, each consisting of the main simulator loop executing for a predetermined number of simulated time steps. This episodic structure is what is used by the neuroevolution algorithm to provide feedback to the neural network controller as to how well its actions did over the course of an episode.

3.3.1 Simulator Loop

The execution of the simulator consists of three key steps, corresponding to the general robotics tenets of sense, process, and execute. Here those steps consist of:

1. Sense state

2. Compute control solution

3. Apply control solution

Items 1 and 3 are interactions between the environment and the agent, so that part of the processing is done by the agent construct, and the rest by the world construct. Namely, once the agent has selected an applied force vector $\vec{T}$, it is up to the world to update the agent acceleration, velocity, and position since the agent does not contain an explicit model of the physics to which it is subjected, but must learn over time how a particular action selection affects its heading and position relative to the sensory data it collects. Figure 1 show the serial execution of the algorithm for a single episode of simulation, beginning at user specified initial conditions and ending at a predetermined run-time.

```
input  : Randomly positioned waypoint
output: Text file charting agent performance
while  T < T_max do
    Sense state s
    if Not at final waypoint then
        if Not at current waypoint then
            Select force vector ν
            Sum agent accelerations
            Update agent velocity η
            Update agent position ρ
        end
        else
            Select next waypoint
        end
    end
    else
        Store network
    end
end
Calculate episode reward R
```

Figure 3.3: Navigation Simulator Control Loop: For simulating a single episode consisting of T_{max} time steps in which the agent senses its state, selects an applied force vector, and using this and approximate Newtonian mechanics its velocity and position are updated.

3.3.2 Artificial Neural Network

An artificial neural network is used in this research as a functional mapping between the agent states (path sensors) and the agent actions (selecting a force vector). As described in Section 5.1, there are nine path sensors that collectively characterize the agent state at each time step of the simulation according to the path length approximation described in Equation 3.1. In order to effectively use this sensor information in the neural network structure, we must eliminate potential scale effects by normalizing the values such that $\mathbf{S} \in [0, 1]$.

To produce the outputs that are used to form the applied force vector $\vec{T}$ these nine input values are fed forward through the neural network, so that each of the thirty-six nodes in the hidden layer is either activated or not from the sum of the inputs multiplied by their weights. Feeding forward continues from the hidden layer to the output layer, yielding three output values that are again between 0 and 1, each corresponding to one of the three cartesian components of $\vec{T}$. In order to use these components as the applied force, they are simply scaled using the maximum available thrust of the aircraft.

So in interacting with the simulator loop, scaled sensory data is used as inputs to the neural network which feeds these values through the network to obtain output values that are scaled into the applied force vector $\vec{T}$.

3.3.3 Neuroevolution Training Loop

To evolve a controller, we begin with an initially random population of thirty networks for our single agent. During each episode, a single network is used over the entire runtime. After each episode, consisting of one simulation over a series of time steps, a reward for the network is calculated as shown below in Equation 3.6.

```
input  : Population of random networks
output: Trained flight controller
i = 0
for  i < Episodes do
    Initialize UAV
    while  T < T_max do
        See Figure 1
    end
    Calculate episode reward R
    Re-rank network population
end
```

Figure 3.4: NeuroevolutionTraining Loop: Used over successive episodes for the training of a neural network controller by iteratively changing the weights between nodes to develop a trained controller for a UAV to fly and navigate.

To train the networks, their performance over an episode must be compared to some value. For the flight and navigation domain, we will use a reward function that is again based on the notion of the approximate path length, so that even if an agent does not get nearer to the waypoint, as long as it turns towards the waypoint it will be rewarded. This is necessary because instances could arise where the waypoint is directly behind the agent and the agent cannot make a zero radius turn. The agent must therefore get further away from the waypoint in order to complete a turn to approach the waypoint.

DIMENSIONS	
α	0.3
β	0.5
γ	0.2
$z_{desired}$	500 (m)
CruiseSpeed	22 $(\frac{m}{s})$
Δ_t	0.1 (s)

Table 3.2: Table of the parameters used in calculating the reward function F.

Using the parameters in Figure 3.3.3, the reward function is calculated as in Equation 3.6. The reward function is composed of three terms that characterize the desired behavior of the agent as we have defined.

1. The first term (Λ_1) is simply the difference between the altitude of the agent and the altitude that we want it to maintain.

2. The second term is the approximate path length **S**.

3. The third term (Λ_2) is the difference between the speed of the agent and the cruise speed that we would like it to maintain.

$$F = -\alpha * |\Lambda_1| - \beta * S - \gamma * |\Lambda_2| \tag{3.6}$$

$$\Lambda_1 = z_{desired} - z_{agent}$$

$$\Lambda_2 = CruiseSpeed * \Delta t * 1.3 - |\bar{v}_p|$$

By rewarding the agent this way, we can evolve a neural network controller that maintains altitude, flies at a speed reasonable for the airframe (which might be determined by such factors as energy efficiency and urgency of mission), and will steer towards and approach the waypoint.

4 Navigation Results

The experiments that follow share some common features. The simulated environment is always the same size, with dimensions as listed in Section 5.1. For each experiment, the waypoint for the agent to reach is located near the top center of the world (1000, 1500, 500), so that when the agent is initialized at the center of the environment (1000, 1000, 500) with a velocity in the (0, 1, 0) direction, the waypoint is directly ahead of it. This puts the waypoint close enough to the agent for the initial experiments that it is easily capable of reaching waypoint in the time allotted. While in some experiments the starting position and velocity of the agent may change, for each experiment the waypoint will be at this position as a common point of reference. Recall that since the state space of the agent relies on *relative* positions of the agent to the waypoint (Section 5.1, changing the position of either the agent or the waypoint has the same effect.

To give some sense of how well the agent is able to achieve this, we decided to implement a simple algorithm that uses a basic feedback mechanism to update its applied force vector from one step to the next, using the same sensory information available to the neural network controller.

While the physical equations governing the system (3.2, 3.3 , and 3.4 are coupled in their x, y, and z components, we can approximate the system to a degree by simply assuming that the coupling is weak. Then the vector components can be separated into multiple single input single output (SISO) systems rather than one multiple input multiple out system (MIMO). Then feedback can be used separately for the z component of the applied force vector, and for the heading, or azimuth, of the vector in the xy plane, from which new the x and y vector components can be found by applying a rotation of θ about the z-axis.

Equations 4.1 describe the precise mathematical operations necessary for this where $\vec{T}$ is the applied force vector with components (T_x, T_y, T_z). The parameters θ and ϕ in these equations are detected from the sensor inputs by performing a search over each sensor value to find the lowest value, which as described in Section 5.1 corresponds to the shortest flight to the waypoint. The angle θ is the zenith angle difference added to the current velocity of the agent, and the angle ϕ is azimuth difference added to the velocity. So for potential paths to the left of the agent θ is positive, to the right it is negative, and for paths upward of the current velocity ϕ is positive, while for those below ϕ is negative.

$$T_x^{t+1} = T_x^t cos(\theta) - T_y^t sin(\theta) \tag{4.1}$$

$$T_y^{t+1} = T_x^t sin(\theta) + T_y^t cos(\theta) \tag{4.2}$$

$$T_z^{t+1} = \pm T_z^t cos(\phi) \tag{4.3}$$

The aim in designing this controller was to create a simple controller that uses the same state space as the neural network controller so that a fair comparison could be made and we could verify that the neural network has learned good behavior. Because the angular rotations performed in Equations 4.1 are based on the direction corresponding to the sensor (or path) with the lowest value, this controller is called the Best Path First algorithm, abbreviated as BPF in the figures.

4.1 Level Flight

For the experiments testing the ability of the controller to achieve level flight, the UAV agent begins at the exact center of the simulated environment, so that its coordinates

are (1000, 1000, 500). The waypoint is set to be at the same elevation as the agent's initial height so that only maintaining initial altitude is necessary for the agent. While flight control and navigation are not completely separable, the most expedient way to test just the flight control is to set a waypoint directly along the initial trajectory and at the same level as the agent. Thus all the agent has to do is to maintain heading and altitude and it will reach the waypoint, maximizing the reward function 3.6.

To determine how well the controller is able to learn, we can look at the system reward value over successive training episodes and notice that the system reward rises. In Figure 4.1 we can see the system performance using both the evolved neural network controller, and the linear controller.

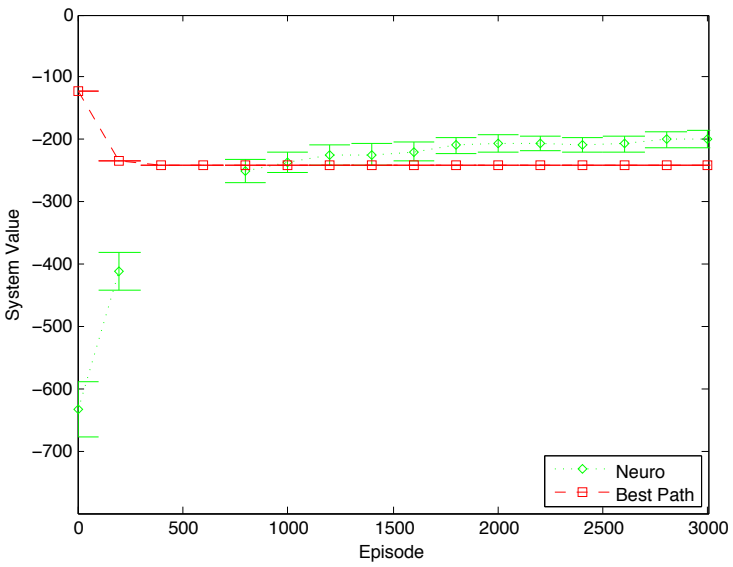


Figure 4.1: Plot of the performance of both the neural network controller and the linear controller for Experiment 1: level flight. Learning of the neural controller over successive episodes is evidenced by the rise in system value over time.

Determining the actual behavior of the controllers is difficult from looking at such curves however, so in an effort to expedite this we have created visualization tools that show the actual position of the agent over the course of a single episode. Each

of these plots is a projection of the three dimensional agent position onto the ground of the simulated environment as trying to visualize this is three dimensions becomes very difficult. For every one of the plots generated this way, the episode chosen for the visualization is one in which the neural network controller has been trained and its reward has converged to some value.

Figure 6.2 shows the x, y position for an agent over an episode, using both the neural network controller and the linear controller. The neural network learns to go straight for the waypoint, while the Best Path First (BPF) controller struggles somewhat as can be seen in the oscillations in its flight path. The oscillations that are present arise largely due to the inability of the BPF controller to account for the coupling between its speed and the resulting lift force due to the simplifying assumptions that were made.

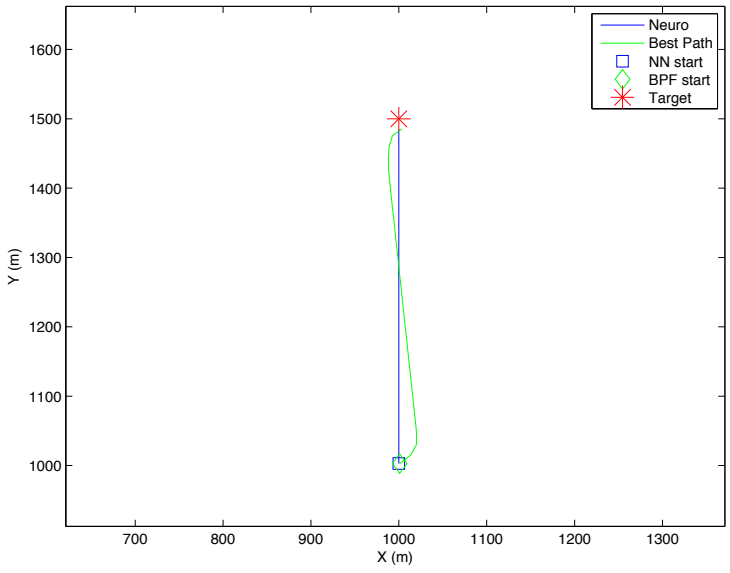


Figure 4.2: Plot of the path of the agent over the course of an episode for Experiment 1: level flight. The path of the neural network controller is shown in blue heading directly for the waypoint, while the Best Path First algorithm has some small oscillations. We see both controllers able to steer the agent towards the waypoint shown in red.

4.2 Reaching a Waypoint

Adding significantly more complexity, we next look at having the agent not only fly without losing altitude, but also having to turn towards a waypoint and ultimately reach that waypoint. Beginning with a simple case and becoming progressively more complex, the first of these experiments consists simply of the agent having to turn approximately 270° towards the waypoint. In Figure 4.2 we see that the controller again converges quickly to a solution, but as it must fly further overall in the time allowed, it converges to a lower value than in the previous experiment.

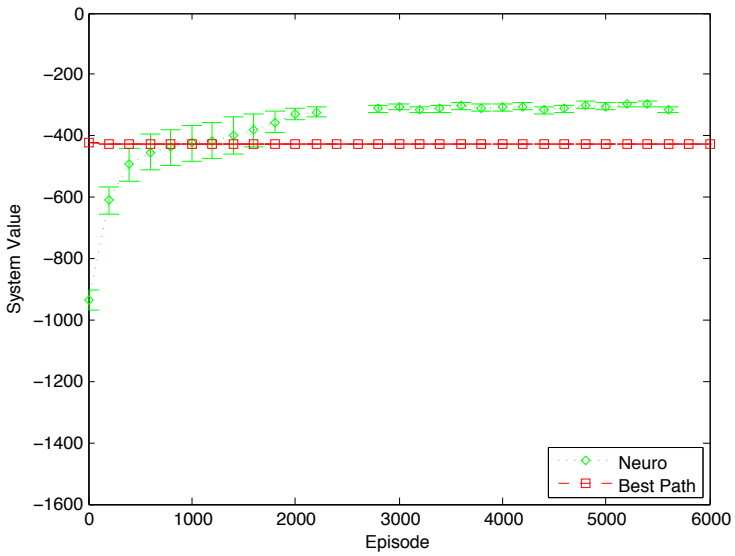


Figure 4.3: Plot of the performance of both the neural network controller and the linear controller for Experiment 2: combining flight control and navigation. For this more complex situation we see the simple BPF controller make a smooth turn, while the neural scheme learns to make a similar smooth turn, preserving its speed and altitude at the sacrifice of some accuracy in reaching the waypoint.

As before, it is helpful to visualize the behavior of the agent over an episode as shown by its change in position relative to the waypoint that it is trying to reach. Figure **??** again shows the flight paths of an agent using both the neural network controller and the BPF controller. The controller using the BPF algorithm smoothly

and quickly reaches the waypoint as expected, while the neural network controller seems to place too much emphasis on maintaining speed and altitude and as such appears on this two-dimensional projection to not come as close to the waypoint as the BPF controller. However, as the BPF algorithm does not account for speed and altitude as well as the neural controller, the system rewards achieved by the two are very similar again.

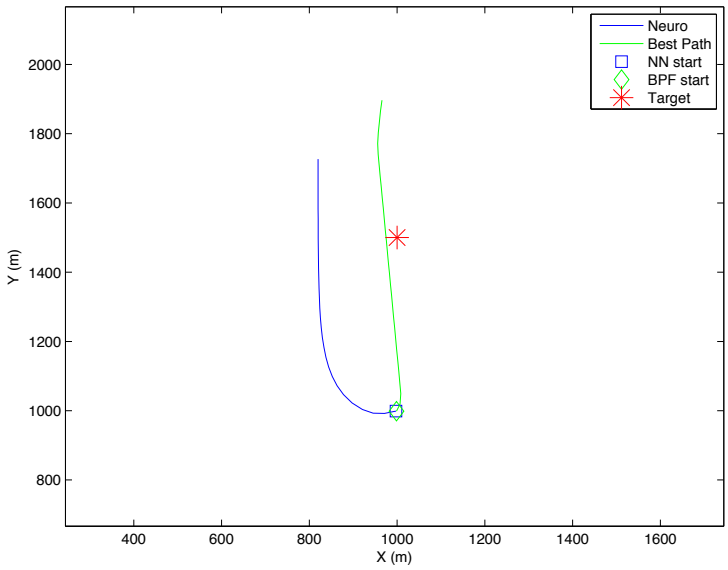


Figure 4.4: Plot of the x,y position of the agent over the course of an episode for Experiment 2: combining flight control and navigation. The neural controller learns to make the turn towards the waypoint with some steady state error, while the BPF controller makes a smooth turn directly to the waypoint, sacrificing some speed and altitude in the process.

Extending the last experiment just a little further, what happens if the agent is facing completely the wrong way from the waypoint? We see in Figure 6.2 that the neural network controller handles this easily, while the BPF controller handles this significantly worse than the previous experiment that required less change in its heading, as it tends to overcorrect its trajectory as it nears the waypoint. However, we can see that as it passes the waypoint, it does begin to correct its heading. This

overcompensation results from a simplistic use of the sensor information that does not distinguish well between small and large changesin sensor values.

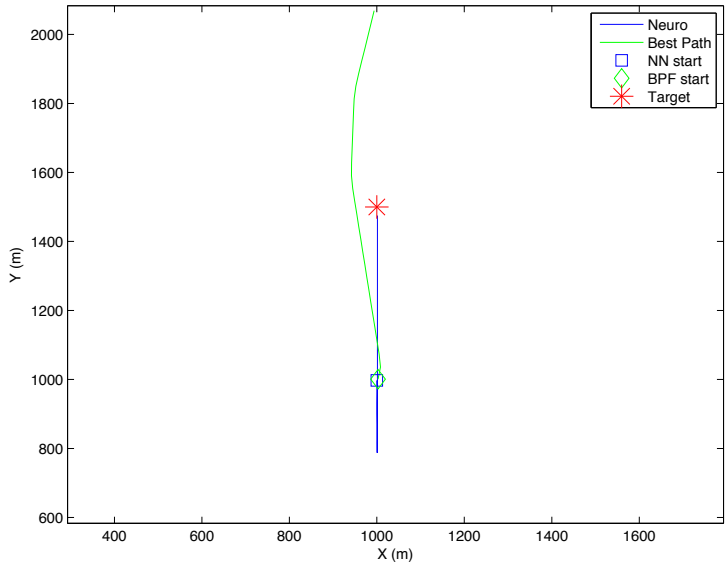


Figure 4.5: Plot of the x,y position of the agent over the course of an episode for Experiment 3: combining flight control and a full heading reversal. With an initial heading leading it directly away from the waypoint we see the neural controller make a complete turn very quickly to reach the waypoint, while the linear controller shows some oscillatory behavior near the waypoint as sensor values change rapidly.

Adding further complexity to the task that must be completed, the next experiment adds a significant dynamic element to the training task by changing the starting position of the agent every 1000 episodes, so that the waypoint is not directly inline with the initial heading of the agent. This forces the neural network controller to use the sensor values for navigation in a similar manner to the linear controller, whereas in the first two experiments, the static nature of the environment meant that it was possible for the agent to simply memorize which way it had to go. To add this extra complexity, we set the starting coordinates of the agent to anywhere in the environment, with the caveat that the height is constrained to between 500 and 1000 m, so that the agent is not doomed to failure by starting too close to the ground and having

an immediate catastrophic failure.

Figure 4.2 shows just one episode after the neural network controller is trained, and while the neural controller and the BPF controller do not start from the same (shown by the symbols on the graph), neither has an initial trajectory that leads it directly to the waypoint so we feel that a fair comparison can be made. The BPF algorithm is able to begin a very nice initial approach to the waypoint despite and initial trajectory heading away from it before it ultimately begins to oscillate and overcorrect its heading as before. The neural controller immediately starts changing its heading to approach the waypoint before it too begins to overcorrect. Note that the scale has changed to smaller gradations so that the behavior can be observed, so neither controller is actually missing by as wide a margin as it might first appear. While this does not show the neural controller converging to a solution that reaches the waypoint quickly and every time, it does show the generalizability of the solution and a robustness to nonlinearities that is not present in the linear controller.

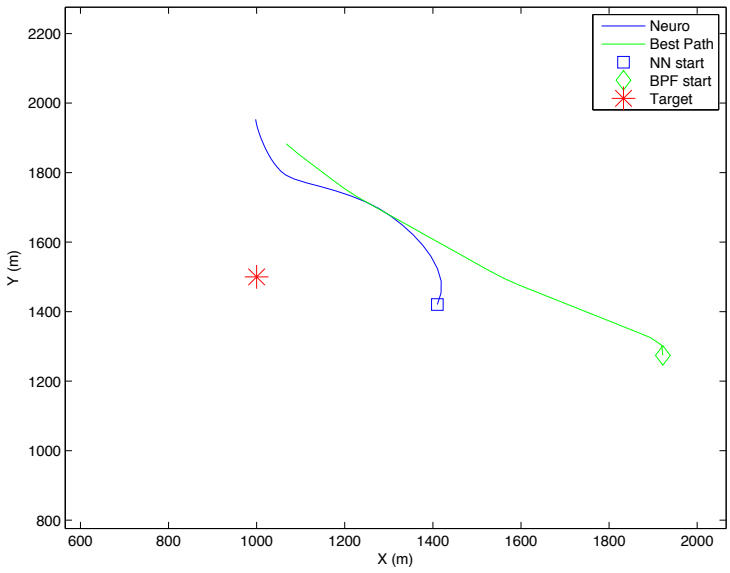


Figure 4.6: Plot of the x,y position of the agent over the course of an episode for Experiment 4: combining flight control and navigation in a dynamic environment. On a finer scale than previous plots, we see here that both the neural network and BPF controllers overcorrect somewhat as they near the waypoint.

4.3 Sensor and Actuator Noise

For a flight controller to ever be applied on a physical platform, it is necessary that it is tolerant of some noise in both the sensors and the actuators, as real robotic systems are always stochastic in nature []. Actuators in particular tend to be particularly problematic due to mechanical slip and other problems, so it is vital that the proposed controller be robust to a significant amount of actuator noise. To determine how affected the evolved controller will be to these types of noise, some noise will be introduced into either the sensor values or the components of the applied force vector.

4.3.1 Sensor Noise

The presence of sensory noise in any real system is possible for any number of reasons including (but not limited to) electrical interference, aliasing, poor resolution, and many others. Therefore, in choosing a controller it is important that such noise does not seriously hamper performance. To evaluate this, we have introduced noise into the simulated sensors used by our agent. The procedure is shown in Figure 3 that modifies the agent's vector of sensor values σ by multiplying each by a factor, giving a modified state vector denoted σ'.

input : σ
output: σ'
for $i < I$ **do**
  $\Omega \in [0, \epsilon]$
  if $rand()$ mod $2 == 0$ **then**
    $\Omega \leftarrow \Omega * -1$
  end
  $\sigma'_i \leftarrow \sigma_i * \Omega$
end

Figure 4.7: Navigation Sensor Noise Algorithm: the routine by which a level a random error is introduced into sensor values.

To test the robustness of the neural controller to sensor noise, we will repeat a previous experiment but add $\pm 5\%$ sensor noise to each sensor. Figure 4.3.1 shows that the neural controller is sensitive to even a small amount of sensor noise. While it does begin to turn towards the waypoint, it fails to make enough of a turn to approach the waypoint. The BPF controller also seems to struggle greatly with this level of noise, as its tortuous path indicates.

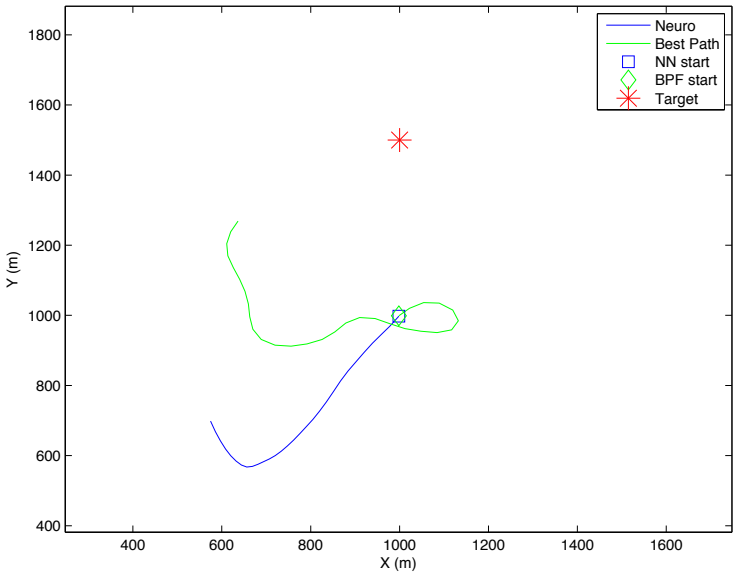


Figure 4.8: Plot of the x,y position of the agent over the course of an episode for Experiment 5: introducing 5% noise into the sensor values. We see that the neural controller struggles to as it initially tries to locate the waypoint before it finally begins to turn towards it, while the highly circuitous route of the BPF algorithm ultimately steers the UAV towards the waypoint but only after a significant amount of maneuvering.

These results indicate that the state space chosen is sensitive to even small amounts of noise, as both controllers struggle with even a noise level of 5%.

4.3.2 Actuator Noise

It is even more important for a controller to be robust to noisy actuators than to sensor noise if the controller is to be applied to an actual robotic platform. To ensure that this controller can handle a reasonable amount of actuator noise, we introduced a large amount of actuator noise (20%) into the system in a fashion nearly identical to that shown for sensor noise in Figure 3.

As with the sensor noise, we again take the second experiment performed (the

full heading reversal as shown in Figure 4.3.2. The neural controller again struggles initially to turn towards the waypoint, and once it does it fails to get completely there and passes close to it without reaching it. The BPF algorithm makes an initial smooth turn towards the waypoint but also overshoots and only after it has passed the waypoint does it begin to correct this.

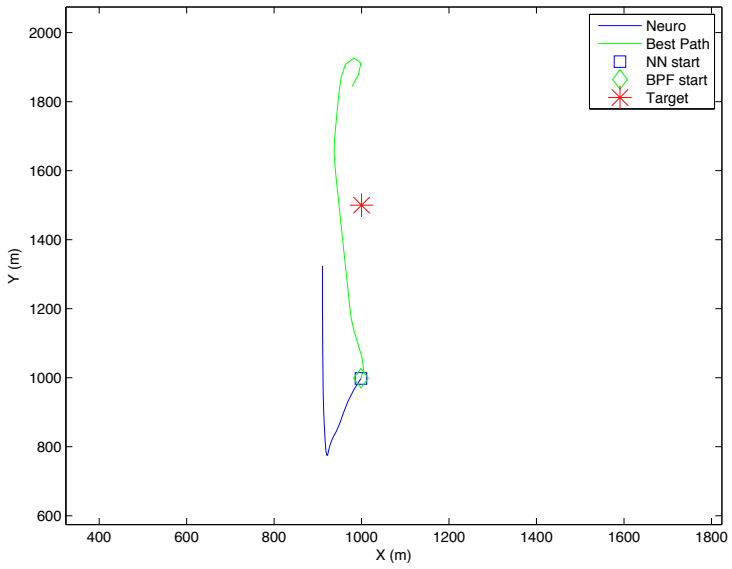


Figure 4.9: Plot of the x,y position of the agent over the course of an episode for Experiment 6: introducing 20% noise into the actuators. Both controllers are ultimately able to turn towards the waypoint, but with significantly different behavior. The neural controller struggles initially before making a turn that leads it pass the waypoint, while the BPF algorithm makes a good initial turn but overshoots and takes awhile to begin to correct its path.

The actuator noise level of 20% used in this experiment is quite large, but both controllers seemed able to handle it quite well especially when compared to the sensor noise, indicating that the action selection of the net applied force vector works well for this domain.

5 Coordination of Autonomous Aircraft

In evolving a neural network controller, coordination among autonomous aerial vehicles can be accomplished in either of two ways. These are a top-down centralized control scheme, or a distributed multi-agent system in which each agent reacts to the actions of the others based on simulated sensors. In selecting agents, the most obvious selection was to have each aircraft be an agent. This was chosen for this research because of a desire to create a controller that could eventually be applied to prototype autonomous aircraft. Sensors collect information about the points of interest (POIs) in the environment and about other aircraft, which is fed through a neural network controller to output navigational waypoints. These waypoints are then used by a simple navigational algorithm which guides the agent to the waypoint.

As with the Navigation algorithm, training of the neural network controller is accomplished through neuroevolution. The algorithm here is the same as with training the navigation controller, including the inputs and outputs. The only significant differences in training are in the population sizes involved and the size of the networks themselves.

5.1 Aircraft Coordination

As mentioned above, the agent selection for this research was an aircraft. The reasons for this are specific to the domain chosen, which is persistent aerial surveillance of an area using multiple UAVs. The specific reasons for this choice are that we want a controller that will generalize to very different arrangements of POIs in the environment, and one that gives a somewhat accurate representation of how the data will actually be collected. For instance, work in the air traffic domain has selected an

	Essential Flight Characteristics	
v_{max}	maximum speed	12 $(\frac{m}{s})$
v_{min}	stall speed	5 $(\frac{m}{s})$
$\dot{\theta}$	turning rate	$(\frac{degrees}{s})$

Table 5.1: Table of the variables effecting the flight characteristics of the aircraft in simulation.

agent as a fix, or navigational waypoint used for air traffic management [48]. This worked well in the air traffic domain because the fixes are directly used for metering air traffic by assigning miles-in-trail for aircraft passing through the fix. Such an approach does not easily lend itself to the aerial surveillance domain chosen for this research as the layout of the POIs in the environment is not known ahead of time, nor are the numbers of agents or the headings they will choose.

The characteristics of the aircraft chosen for the simulation were modeled after a generic fixed-wing MAV. The flight characteristics of the aircraft deemed most critical for simulating the flight dynamics are the maximum speed, stall speed and are summarized in Table 5.1.

5.1.1 Sensors

Each agent collects information about the environment around it which describes its current state. In order to construct a meaningful representation of the world that effects the agent, information must be collected about both the relative location of the POIs, and the relative positions of the other agents in the system. Obviously, if only a single agent is present in the system this would simplify to only the information regarding the POIs. This makes two sets of sensors for each agent. The resolution of

these sensors depends on how we choose to discretize the spatial orientation of each set of sensors. In order to make this choice somewhat realistic, the geometry of the sensors was loosely modeled on sonar sensors, so that each sensor has a detection area of some number of degrees, and the total sensor area is a circle centered about the agent as depicted in Figure 5.1.1. As shown, for each set of sensors, the total sensor area is subdivided into five equal regions about the agent, each corresponding to a sensor. Thus, there are a total of ten sensors, the values of which quantify the state of the agent and serve as inputs to the neural network.

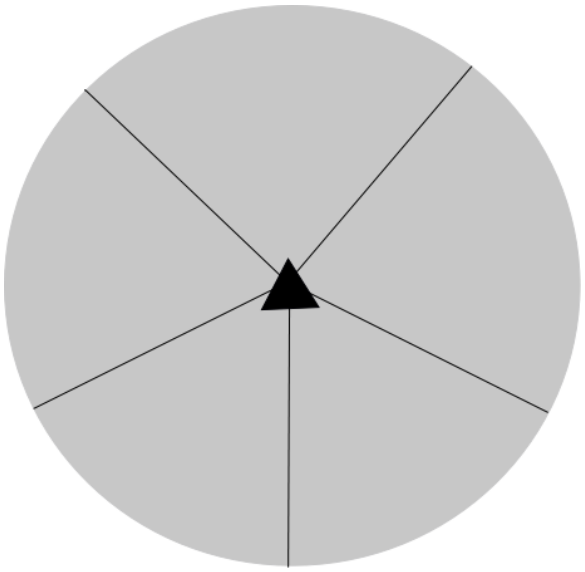


Figure 5.1: Sensory setup for the agent consisting of five equal regions in a circle centered about the agent. Two of these sensor arrays are used: one for other agents in the system and a second for the POIs. Each sensor value is just a sum over all of the agents or POIs in that region scaled by their approximate flight path length from the agent as in Equation 3.1.

A value (λ_{ip}) for each POI sensor i is calculated based on the number of POIs in that region, while for each agent j sensor, a value (λ_{ja}) is calculated based on the number of agents excluding the agent sensing. Using the approximate path length S' as calculated in Equation 3.1, the numerical value for each POI sensor is the sum

over every POI in that region, of the value of the POI divided by the approximate path length S' to that POI as in Equation 5.1.

$$\lambda_{ip} = \sum_{n=1}^{N} \frac{V_n}{S'_n} \tag{5.1}$$

Similarly, the numerical value for each agent sensor is the sum over every agent in that region, of the value of the agent divided by the approximate path length S' to that agent as in Equation 5.2.

$$\lambda_{ja} = \sum_{m=1}^{M} \frac{V_m}{S'_m} \tag{5.2}$$

5.1.2 Action Set

After feeding the ten sensor values to the neural network controller, the spatial coordinates for a navigational waypoint are output. The range of possible coordinates are constrained to an area bounded by two circles, each of which represent a logical bound on where the aircraft can feasibly go based on its current heading and speed. The innermost circle represents the area that cannot be easily reached if a waypoint was selected that is behind the agent due to its forward momentum, while the outer circle represents a bound based on the speed of the aircraft. As depicted in Figure 5.1.2, this region obviously only approximates the region that can be reached easily by the UAV. The radii of the circles (ρ_{inner} and ρ_{outer}) were tuned by hand to some fraction of the total size of the environment.

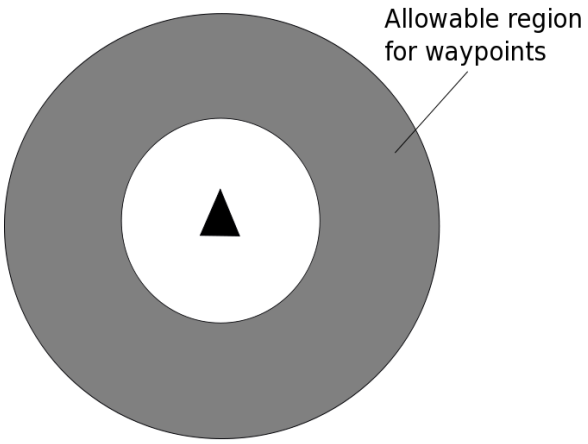


Figure 5.2: The grey toroidal area depicts the bounded region in which valid waypoints may be selected for the aircraft when one is not already selected and being approached. By limiting where waypoints can be chosen, we can speed up learning by ensuring feasibility of solutions.

Bounding the region in which valid waypoints may be selected ensures that learning will converge to reasonable solutions quickly rather than struggling with so many different possible options that learning can only occur given an exceedingly large number of training episodes, if at all.

5.2 Simulator

The main purpose of setting these dimensions is not to constrain the motion of the agents, but to serve as a guide in the initial distributions of both the agents and POIs, which are described in more detail for individual experiments in Chapter 6. The physical dimensions of the simulated environment are:

- Width = 5000 (m)
- Height = 5000 (m)
- Depth = 3000 (m)

Obviously, these dimensions are somewhat arbitrary, but they were chosen to work well with the maximum speed of the aircraft agents in the sense of simulating a large enough world for the amount of time used for each training episode.

The coordination simulator was written for the specific purpose of being able to have many agents evolving concurrently, which can be computationally expensive depending on the number of agents, the size of the network population that each is evolving, and the size and complexity of the networks that comprise the population. With this in mind, the flight dynamics for this simulator have been simplified computationally so that reasonably short simulations can be run for effective experimentation. Therefore, the physics governing flight are not modeled in this simulation, but are instead approximated by using realistic values for the turning rate ($\dot{\theta}$) of the aircraft.

5.3 Algorithms

To create a controller for individual agents using a model-free approach, the controller must evolve over the course of many episodes using feedback from a reward function. The value of the reward function is calculated for an episode which consists of a controller taking multiple actions over time, so that at the end of every episode, the controller used is evaluated over the entire episode. While such training could conceivably occur by creating prototype aircraft and using the controller to fly them, the cost and time taken would be prohibitive compared to training the controller in a simulated environment. In this section we will discuss the simulator used for training and why it can be considered a good alternative to training on physical systems, and how the controller as modeled by an artificial neural network is actually trained.

5.3.1 Simulator Loop

Training of the neural network controller depends on both the main simulator control loop, and on the neuroevolution loop. An episode consists of an amount of time that elapses in discrete time steps, and at each time step if the agent is not already busy flying to a waypoint, the agent reads in sensory data and sends it to the neural network which calculates the action to take in the form of a navigational waypoint to approach. At time steps where the agent is already busy flying to within some small distance of the current waypoint, the agent velocity and position are updated according to the approximate flight dynamics model used.

```
input  : Vector of sensory data
output: Navigational waypoint
while  T < T_max do
    for  each agent i ∈ I do
        Sense state s
        if Waypoint reached then
            Select new waypoint ω
        end
        Update agent position ρ
    end
end
Calculate episode reward R
```

Figure 5.3: Coordination Simulator Control Loop: Used to provide feedback to a controller for coordinating multiple UAVs in persistent aerial surveillance. Over a series of discrete time steps, the UAV senses its state, selects an action, and this is used to update its velocity and position.

The specific flight dynamics used in the simulator do not calculate external forces and sum them at each time step as done for the flight control and navigation simulator. Instead, if the agent is not within some tolerance of the waypoint, and if the agent is not directly facing the waypoint to within some tolerance, then the agent velocity changes direction towards the waypoint by an amount equal to the turn rate of the

aircraft. After this, all that remains is to update the agent position $\bar{R}$ using simple Eulerian numerical integration according to Equation 5.3.

$$\bar{R}^{t+1} = \bar{R}^{t} + \bar{v}^{t+1} * \Delta t \tag{5.3}$$

5.3.2 Artificial Neural Network

The sensors as described in Section 5.1 serve as the basis for the inputs to the artificial neural network controller. Because the inputs can range from zero to infinity, they must first be scaled to between zero and one so that the neural network weights are not artificially skewed by sensor values that are higher than normal in certain scenarios. This simple scaling operation involves dividing each sensor value by the maximum sensor value.

As before, the weights of the neural network will be mutated by the neuroevolution algorithm and can range in value from zero to one, with a weight of zero meaning that data from the associated node is not used at all. The activation function used in the hidden layer nodes of the network is a sigmoidal activation function so that discontinuities do not occur as they might if a simple step function was used.

As indicated in Section 5.1.2, there are ten inputs to the neural network, each representing a sensor input from either POIs or other agents. The neural network has two outputs representing the x and y coordinates of a navigational waypoint for the agent to reach. With a hidden layer comprised of 30 nodes, there is sufficient network complexity to provide a good mapping of inputs to outputs.

5.3.3 Neuroevolution Training Loop

To train the networks, the reward that a network is given can be based on the system performance, the performance of the individual agent, or some combination of the two. Here we will denote the system performance as G, the individual or local performance as L, and the combination of these as the difference reward D. The simplest of these to define is L, which is just the sum over all the POIs of the value of the POI divided by its approximate path length S'. G is the most domain specific of the objective function to define, as it depends the most on what the designer wants the agents to do in the environment.

$$L = \sum_{m=1}^{M} \frac{V_m}{\left|\vec{R}_{i,m}\right|} \tag{5.4}$$

We will define G as the sum over all of the POIs of the value of the POI (V_m) divided by its euclidean distance for only the closest agent.

$$G = \sum_{t} \sum_{j} \frac{V_m}{min_i(R_{m_t,j})} \tag{5.5}$$

In using G to reward individual agents, we are giving the an objective that is closely aligned with the overall objective, ensuring that when they improve their own performance they are also improving the system performance. We will call this quality the *factoredness* of the reward function [1]. As the number of agents in the system increases, the effect of the individual agent on the total system reward becomes less discernible, essentially creating noise for the reward signal. This quality of being able to distinguish the agent reward from the system level performance is dubbed the *learnability* of the reward function [1]. Thus, in terms of the two criteria listed, G has high factoredness but low learnability as an agent reward. In contrast to G,

the reward for the individual agent will have high learnability as there are no other agent actions being taken into account in the reward function, but the factoredness is low because as defined in Equation 5.5 only the agent closest to each POI actually contributes to G.

To create a reward function that possesses both high factoredness and high learnability, reward shaping must be used. To this end, we will define D as a combination of G and L such that we are able to isolate the effect that the agent has on the system [1]. The general form of D can be seen in Equation 5.6, consisting of two terms. The first term is just G of the entire vector of agents $\bar{z}$, while the second term is composed of G of all the agents except the agent for which D is being calculated, while adding an additional term c_i that can be defined in many different ways.

$$D_i = G(\bar{z}) - G(\bar{z} - \bar{z}_i + c_i) \tag{5.6}$$

To tailor this equation for the persistent aerial surveillance domain, we will choose c_i to be equal to zero, so that D simplifies down as much as possible. It is possible to choose c_i to be some other constant such as the average of all the possible actions that the agent could have chosen from its current state, but because the training here is taking place over many different time steps, each of which results in an action by the agent, this average becomes difficult to define. Thus, the final form of D used for training becomes Equation 5.7.

$$D_i = G(\bar{z}) - G(\bar{z} - \bar{z}_i) \tag{5.7}$$

The training loop used is for the coordination controller identical to that used for training the flight control and navigation as shown in 2 in Chapter 3.

6 Coordination Results

To determine the effectiveness of evolving a coordination controller for the persistent aerial surveillance task, it was necessary to conduct certain experiments to evaluate the desired properties of the controller. The first step after constructing the simulated environment was to choose which experiments would be performed and the values of free parameters to be chosen for each, such as the number of agents, the number of POIs, and less well defined: how dynamic to make the environment.

The agents are initially arrayed in a region of the simulated environment as depicted in Figure 6, and the POIs begin with positions in a circle of radius equal to half the width about the center of the environment. This gives a simple starting condition in which even a random control policy can be expected to do quite well. The agents can have any initial heading but no vertical component to the velocity, and each starts with a standard cruise-speed as the magnitude of their velocity.

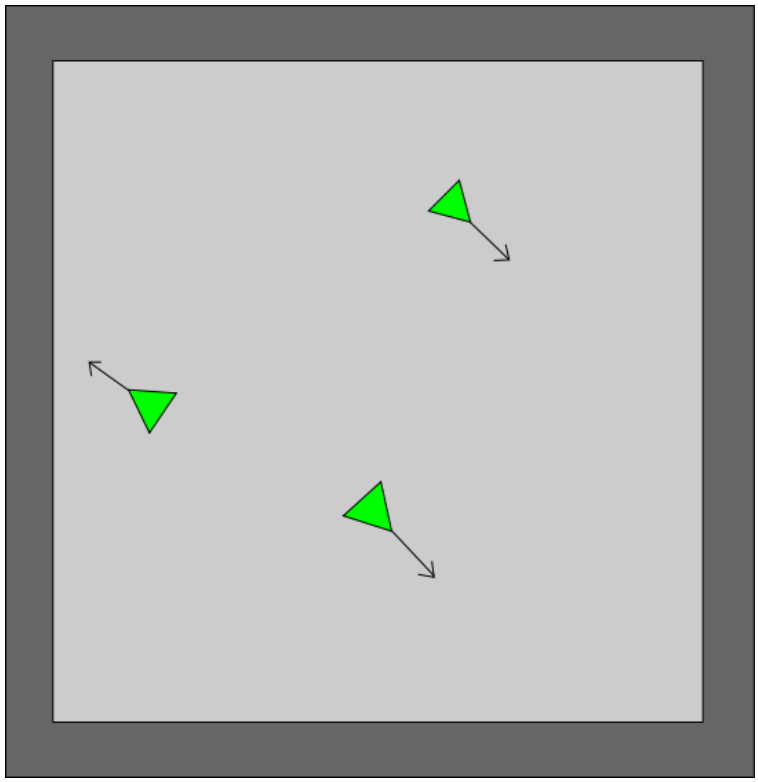


Figure 6.1: The light grey area shows the region in which agents can have initial positions at the start of an episode. This spans 80% of the width and 80% of the height of the environment.

6.1 Static Environment

For this domain with multiple agents that do not communicate, an agent is essentially treating other agents as objects within the environment, so for any system with more than a single agent, even with static POIs the environment is still essentially dynamic in nature. A static environment is somewhat uninteresting by itself, but it allows us to investigate aspects of the system dynamics that may be somewhat obfuscated in more dynamic environments. Such factors include reward structure, scalability of the controller, and the size of the available action set, as depicted in Figure 5.1.2 where we can determine the effect of changing the radii of the inner and outer circles.

Each experiment shows the data collected for a system of 24 POIs and 3 agents, each with a starting population size of 20 networks. Each agent is using the total system reward G to train its population of networks. For comparison, a controller that chooses a random x and y waypoint location in the reasonable bounds is included. Each series of simulations is repeated 20 times so that there is statistical significance to the data which is clearly seen in the relative size of the error bars to the signal.

It can be seen in Figure 6.1 that the controller evolved using G easily outperforms the random controller, and D only manages to match it with so few agents in the system. This indicates that even though the random controller can go in nearly any direction and have agents find POIs, because of the way agents are rewarded cooperation between agents, or at least a self-interested desire to avoid competition is necessary to achieve good system performance.

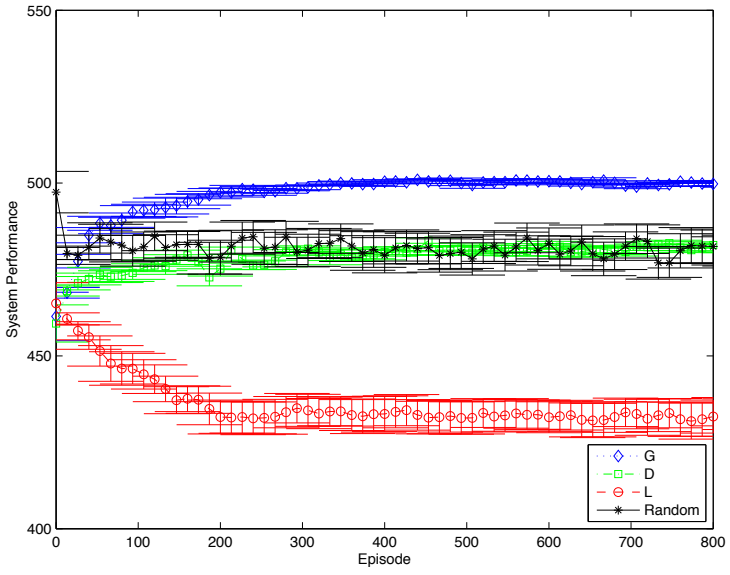


Figure 6.2: Plot of system value over time for 3 agents where agents are rewarded using G, D, L, or a random policy is used. We see that L has learned behavior that lowers the overall system value while G maximizes it. The random policy does well here because of the initial configuration of POIs surrounding the agents initial position.

We can easily see why even the random control policy does so well if we look at visualizations similar to those presented in Chapter 4 for the flight and navigation of a single agent. Figure 6.1 shows the arrangement of the POIs encircling the agents and the positions over time of the three agents trained using G as a reward function.

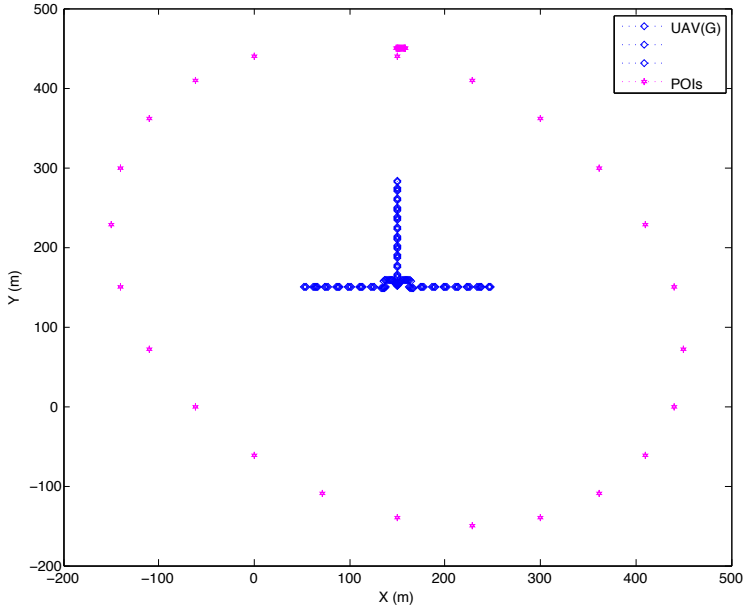


Figure 6.3: Positions over time of the 3 agents using G as they begin from the center of the environment, surrounded by POIs in a circle around the edge of the plot.

In looking at this Figure 6.1 we might actually expect the random policy to perform nearly equal agents using G, but looking at the trajectories of the agents using a random policy in Figure **??** we see that these agents are constantly getting in each others way despite the abundance of value around them.

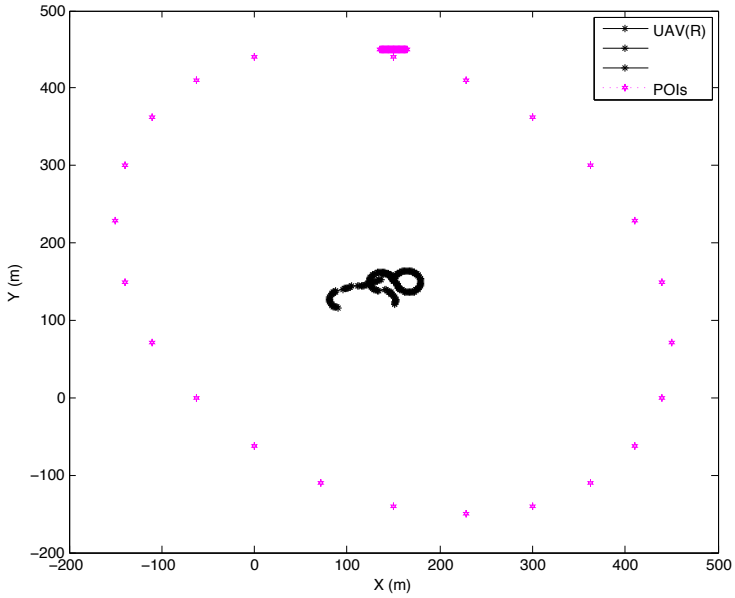


Figure 6.4: Positions over time of the 3 agents using a random control policy as they begin from the center of the environment, surrounded by POIs in a circle around the edge of the plot.

To investigate the scalability of a system using the evolved controllers from the three types of reward signal described in Section 5.3.3, the number of agents is increased from 3 up to 30, and the number of POIs is increased from 24 to 120. Figure 6.1 shows the progression of the system level reward G for each of the three agent rewards. As the number of agents has increased here, the overall value of G has also increased, so it is not the absolute value of G that is of interest here, but rather the relative values of the three different reward structures. The reward using only the performance of a single agent (L) as a metric for evolving the controller not only does poorly, but as shown by its decrease over several hundred episodes has actually learned behavior that significantly lowers the value of the system. This is expected here because L is not factored with G, and for 30 agents coordination must occur to produce a good system level reward. The agents using G do seem to be learning, but it is occurring so slowly (Figure 6.1) that by the end of the trial only marginal im-

provement has been made from the initial value of G. Again, we expect this because as an agent reward signal for a system of 30 agents, G has low learnability. D on the other hand has done significantly better than either L of G, because it has high factoredness and high learnability.

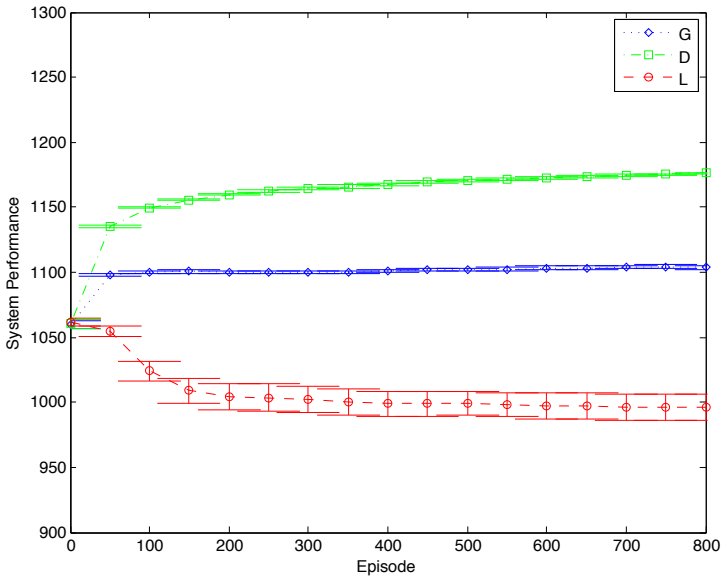


Figure 6.5: With a system of 30 agents, the differences in value of the different reward structures become markedly apparent. The difference reward D is superior to G and L as an agent reward because of its high factoredness and high learnability, while the local reward L seems to learn the wrong behavior.

Because of the very tendency of agents using the local L reward to not work well in an environment with many other agents, we will stop considering this a viable alternative for a coordination controller, as any controller we design may eventually be needed to scale up significantly depending on the task.

6.2 Mobile POIs

By allowing the POIs to move within the environment in addition to the agents themselves, the environment becomes significantly more dynamic, and thus a more difficult problem to solve. Because the domain specified for this research is persistent aerial surveillance, we decided that the most interesting way to move the POIs would be to have them move as a hiker or enemy jeep might. That is, the POIs do not jump from one point in the environment to another, but at each time step in the simulation they move a discrete distance from their current position in some direction. As the agents have no capacity to model how the POIs might move, we decided that for simplicity of simulation they would move in circles of varying radii. So with a large enough radius the POIs may appear to be moving in a nearly straight line.

Figure 6.2 shows the trajectories of the agents and the POIs over the course of a single episode after the controller has been trained using the global reward G. Here again the agents are able to quickly and neatly disperse themselves so as to take full advantage of their surroundings, corresponding to each agent observing several POIs with no wasteful overlap with other agents.

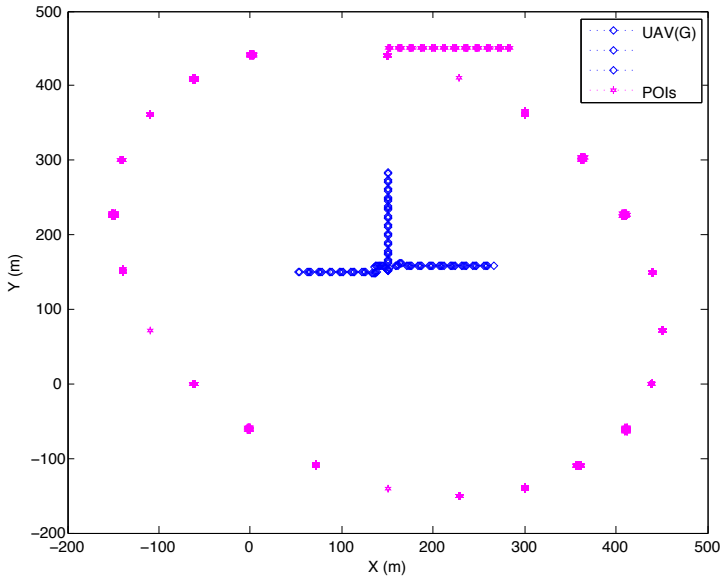


Figure 6.6: Position plot of 3 agents using G tracking mobile POIs. Again, the UAV agents begin in the center of a ring of POIs which can then move from their initial positions.

For this relatively small increase in the dynamic nature of the POIs the system reward is virtually identical to the previous experiment, but the next experiment begins to show more complicated behavior. While POIs moving around slowly in the environment has easy analogies to surveilling herd migrations or something else easy to observe and follow, some things are much more difficult to track. In Chapter 1 while introducing the domain of persistent aerial surveillance we mentioned the scenario of a group of UAVs coordinating to locate missing hikers. One of the most challenging aspects of tracking something small through rough terrain with extensive canopy is that such subjects may seem to disappear and reappear somewhere else.

For the next test of our coordination controller, we look at what happens for POIs that not only move in circles of varying sizes in the environment, but also at what happens when all at once every POI in the system suddenly changes location relative to the agents tracking them. For this experiment, we allow the agents to train for

1000 episodes before completely changing their environment by shifting all of the POIs to a significant distance from their original positions. Looking at the plot of positions for the agents and POIs after the shift in Figure 6.2, we see that the POIs are no longer encircling the agents, but that the agents are now on one side of the circle of POIs.

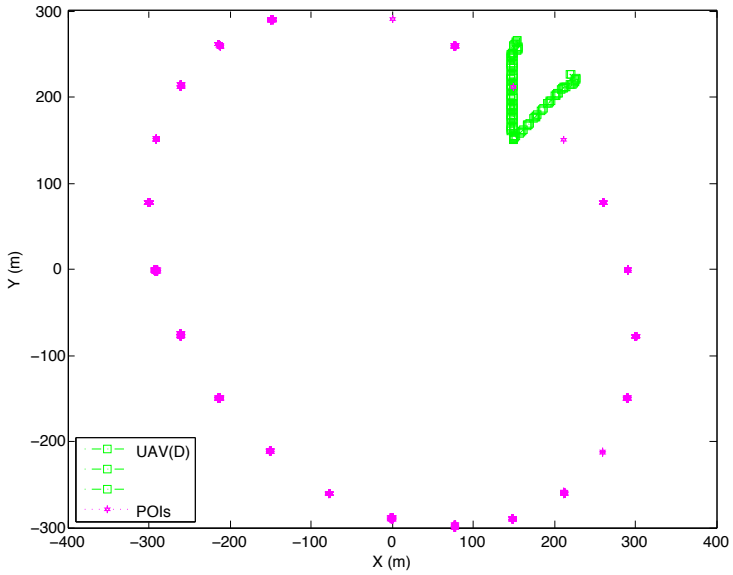


Figure 6.7: Position plot of POIs and agents after POIs have drastically and suddenly changed their positions in the environment. The agents are no longer initially located within the center of the ring of POIs, so must fly further in the same amount of time as before if they are to reach the same level of reward.

Looking at the system performance in Figure 6.2, we see that the performance drops precipitously at 1000 episodes when the sea change of POI positions occurs, but that the agents are able to recover somewhat by beginning to spread back out to observe a larger proportion of the POIs. We see that for this small system of only three agents, those trained using the difference reward D gain no real advantage in the added learnability of the reward signal as there is so little noise in G for such a small system.

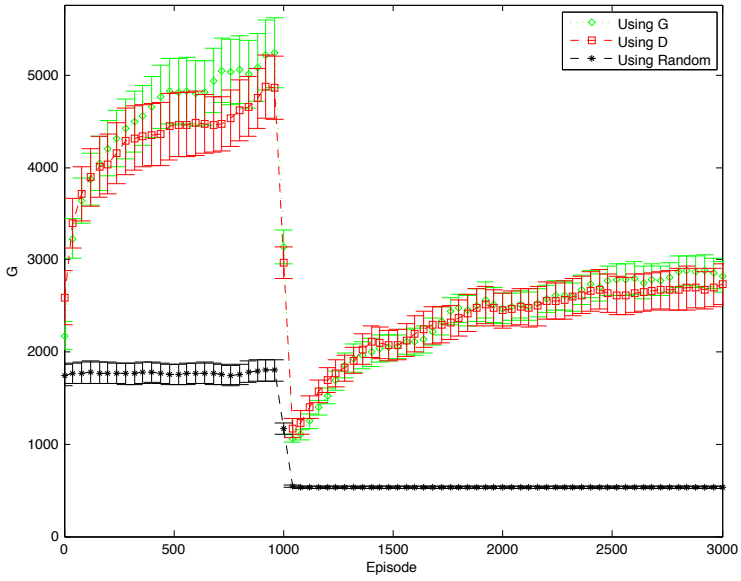


Figure 6.8: Graph showing the huge drop in system value after all system POIs suddenly change positions by a large amount, corresponding to the circle of POIs shifting its center to a point well away from the UAV initial positions.

6.3 Agent Failure

One of the most useful properties of the multiagent systems approach is the robustness to failures by some of the agents in the system. To test how robust to failure our evolved neural network controller is, we recreate a previous experiment in which the POIs were mobile but did not drastically shift positions in the environment. Partway through the training cycle, we disable one quarter of the agents in a system of twelve agents tracking 36 POIs.

Figure 6.3 shows the effect of this sudden failure on the system performance. A small portion of the system reward is lost all at once at 500 episodes, and the system is able to learn to cope with the loss and recover somewhat for both agents using G and agents using D.

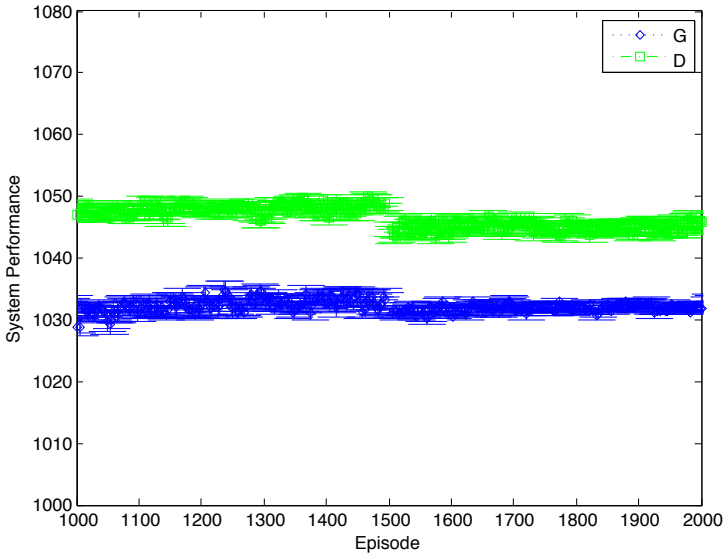


Figure 6.9: Plot of system performance for an experiment in which a full quarter of the agents in the system are suddenly disabled at 500 episodes into the training cycle. We see that with the distributed nature of the system very little impact is felt, particularly using the difference reward D for agent control.

7 Conclusions

A great deal of work has already been done in the field of terrestrial robotics that uses a distributed, multi-agent systems approach. In these domains, though many of the system dynamics are quite different from those of flying aircraft, the similarities mean that the work done here applies quite well, particularly in the coordination of autonomous agents.

As detailed in Chapter 3, the sensory data that serve as the inputs to the flight and navigation controller consist of nine potential trajectories surrounding the current trajectory of the aircraft. Each of these paths is quantified by calculating an approximation of the path length from the current position of the agent to the agent's current waypoint using the agent's velocity vector and relative position to the waypoint. To map these inputs to an action space consisting of a force vector to apply to the agent, a neural network is evolved over successive episodes of simulation by comparing the performance of each network to a function value that relays data about the state of the agent in the environment (see Equation 3.6).

The agent was trained in this fashion for several different scenarios of increasing complexity, and for each of these we evaluated how well the agent was able to learn to perform the task or tasks required. First, we wanted to make sure that the agent could learn to maintain altitude and heading, which was shown by placing the agent's waypoint directly along the initial heading of the agent. The controller quickly learned to only adjust the z component of the force vector to maintain altitude and ignore the other components. The second task was to force the agent to learn to steer towards the waypoint from any orientation. This was done by initializing the agent's position in a random spot in the environment for each episode of training, while leaving the waypoint in the same absolute position, so that the relative positions and the angle

between the velocity and the relative position is different. Again, the agent quickly learns to steer towards the waypoint while maintaining altitude.

From this point we moved from flight control and navigation of a single agent, to developing a controller that coordinates agents for the persistent aerial surveillance domain, in which the aircraft agents try to keep as many environmental points of interest (POIs) in sensor range at any given time as possible. To ensure cooperation and efficiency, only the agent closest to a waypoint is given credit for the surveillance task according to Equation 5.5. Alternate credit assignment schemes are also explored including the local and difference schemes [1, 2, 3, 4, 47, 23, 24].

7.1 Contributions

The research presented in this thesis has been inspired by research in a similar domain. For terrestrial rovers a neuroevolutionary approach has been used for both navigation of an individual rover [19] and for emergent coordination of multiple rovers **??**. The largest distinction between that research and this research is the vastly different dynamics of the agents with the environment, as they have significant physical constraints placed on them including a nonzero turning radius that depends on the speed of the agent, and a minimum speed that must be maintained to avoid stall.

The selection of the sensory and action sets for a single agent in developing a flight and navigation controller are quite novel and designed specifically to allow such a controller to be easily constructed for aircraft with widely different flight characteristics With a state space setup that analyzes nine paths surrounding the current velocity vector of the agent and quantifying each with an approximation of the flight path 3.1, the agent state is succinctly qualified with information including both distance to the target and its relative motion. The selection of the action as

the combination of all applied forces from the control surfaces and impellers of the aircraft into one vector $\vec{T}$ means that this controller could theoretically be modified for use with any type of aircraft.

A coordination controller was developed using a behavioral simulator that realistically captured the system dynamics or multiple interacting UAVs without the added computational burden of simulating the physics for each agent at every time step. Using two types of sensor data for each of five sectors surrounding the agent, information is collected about the other agents in the system and the points of interest (POIs) in the environment around the agent. This information is crystallized into two values for each sector using the path length approximation in Equation 3.1, one value for the other UAVs and one for the POIs.

These values are then used as inputs to a neural network which is evolved over time using either the global system reward G, the local agent reward L, or the difference reward D that is both easy for agents to learn on and also allows actions that positively affect the agent to have a similar effect on overall system performance.

7.2 Future Work

In the flight and navigation domain, future work will necessary lead to additional complexity in the representing a realistic world. This means the agent must be able to sense obstructions along each potential path and learn to avoid these. In theory this can be done simply by the addition of a weighted penalty term to the value of each of the potential paths surrounding the agent's current trajectory. Also necessary in adding complexity to the controller would be some way to detect and adjust for sustained or gusting winds, as these are present in any real environment and can have drastic influences on smaller airframes such as MAVs. These adjustments to the

simulator and sensor data would be enough to be able to put the controller on an actual aircraft, which must be the ultimate goal as simulated aircraft aren't terribly useful by themselves.

For the coordination controller, the most interesting extensions to this work would be to develop new system reward functions for other types of missions that could be encountered by teams of UAVs. Such missions could include terrain mapping, combat, or supply delivery. Perhaps even more important would be research into hierarchical control methods that can determine what type of controller to use in what situation, without the need for user input.

APPENDICES

Bibliography

[1] A. Agogino and K. Tumer. Efficient evaluation functions for multi-rover systems. In *The Genetic and Evolutionary Computation Conference*, 2004.

[2] A. Agogino and K. Tumer. Distributed evaluation functions for fault tolerant multi rover systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Seattle, WA, July 2006.

[3] A. Agogino and K. Tumer. Evolving distributed agents for managing air traffic. In *Proceedings of the Genetic and Evolutionary Computation Conference*, London, UK, July 2007.

[4] A. Agogino and K. Tumer. Efficient evaluation functions for evolving coordination. *Evolutionary Computation*, 16(2):257–288, 2008.

[5] Richard Arning and Stefan Sassen. Flight control of micro aerial vehicles. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2004.

[6] J. Bellingham, M. Tillerson, A. Richards, and J. How. Multi-task allocation and path planning for cooperating uavs. In *Proceedings of the Conference on Cooperative Control and Optimization*, pages 1 – 19, 2001. Mixed integer linear programming optimization.

[7] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[8] P. Chandler, S. Rasmussen, and M. Pachter. Uav cooperative path planning. In *AIAA Guidance Navigation and Control Conference (2000)*, 2000.

[9] A. El-Fakdi, M. Carreras, and N. Palomeras. Autonomous underwater vehicle control using reinforcement learning policy search methods. *Ocean*, 2:793–798, June 2005.

[10] Jose A. Fernandez-Leon, Gerardo G. Acosta, and Miguel A. Mayosky. Behavioral control through evolutionary neurocontrollers for autonomous mobile robot navigation. *Robotics and Autonomous Systems*, 57(4):411–419, April 2009.

[11] D. Floreano and F. Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *Proc. of Conf. on Simulation of Adaptive Behavior*, pages 421–430, 1994.

[12] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 26(3):396–407, 1996.

[13] F. Gomez and R. Miikkulainen. Active guidance for a finless rocket through neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Chicago, Illinois, 2003.

[14] Jeffrey Herbert, David Jacques, Michael Novy, and Meir Pachter. Cooperative control of uavs. In *AIAA Guidance, Navigation, and Control Conference, 2001*, 2001.

[15] J. How, E. King, and Y. Kuwata. Flight demonstrations of cooperative control for uav teams. In *AIAA 3rd "Unmanned Unlimited" Technical Conference*, 2004.

[16] M. Jun and R. D'Andrea. *Path planning for unmanned aerial vehicles in uncertain and adversarial environments*, chapter 6, pages 95–111. Kluwer, 2002.

[17] I. Kaminer, O. Yakimenko, V. Dobrokhodov, A. Pascoal, N. Hovakimyan, C. Cao, A. Young, and V. Patel. Coordinated path following for time-critical missions of multiple uavs via l1 adaptive output feedback controllers. In *AIAA Guidance, Navigation and Control Conference*, 2007.

[18] M. Knudson. *Navigation and Coordination of Autonomous Mobile Robots with Limited Resources*. PhD thesis, Oregon State University, 2009.

[19] M. Knudson and K. Tumer. Towards coordinating autonomous robots for exploration in dynamic environments. In *Proceedings of the Conference on Artificial Neural Networks in Engineering*, pages 587–594, 2008.

[20] Kurt Konolige, Dieter Fox, Charlie Ortiz, and Andrew Agno. Centibots: Very large scale distributed robotic teams. *Experimental Robotics IX: Springer Tracts in Advanced Robotics*, 21:131–140, 2006.

[21] Roman Krashanitsa, George Platanitis, Bill Silin, and Sergey Shkarayev. Aerodynamics and controls design for autonomous micro air vehicles. In *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, Keystone, Colorado, August 2006.

[22] Y. Kuwata. Real-time trajectory for unmanned aerial vehicles using receding horizon control. Master's thesis, Massachusetts Institute of Technology, 2003.

[23] A. Laud and G. DeJong. Reinforcement learning and shaping: Encouraging intended behaviors. In *Int. Conference on Machine Learning*, pages 355–362, 2002.

[24] A. Laud and G. DeJong. The influence of reward on the speed of reinforcement learning: An analysis of shaping. In *Int. Conference on Machine Learning*, 2003.

[25] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 2000.

[26] T. McLain, P. Chandler, and M. Pachter. A decomposition strategy for optimal coordination of unmanned air vehicles. In *American Control Conference, 2000*, volume 1, pages 369–373, 2000.

[27] Manfred Morari and Jay Lee. Model predictive control: past, present and future. *Computers & Chemical Engineering*, 23:667–682, May 1999.

[28] A. Ng, A. Coates, M. Diel, V. Ganapathi, and J. Schulte. Inverted autonomous helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*, 2004.

[29] N. Nigam and I. Kroo. Persistent surveillance using multiple unmanned air vehicles. In *Aerospace Conference, 2008 IEEE*, pages 1 – 14, 2008.

[30] Nikhil Nigam, Stefan Bieniawski, Ilan Kroo, and John Vian. Control of multiple uavs for persistent surveillance: Algorithm description and hardware demonstration. In *AIAA Infotech@Aerospace Conference*, 2009.

[31] Nikhil Nigam and Ilan Kroo. Control and design of multiple unmanned air vehicles for persistent surveillance. In *AIAA-2008-5913, 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2008.

[32] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. How to evolve autonomous robots: Different approaches in evolutionary robotics. In *Proc. of Artificial Life IV*, pages 190–197, 2004.

[33] Reza Olfati-Saber, William B. Dunbar, and Richard M. Murray. Cooperative control of multi-vehicle systems using cost graphs and optimization. In *Proceedings of the American Control Conference*, pages 1–15, 2003.

[34] Reza Olfati-Saber and Richard M. Murray. Distributed structural stabilization and tracking for formations of dynamic multi-agents. In *Proceedings of the 41st IEEE Conference on Decision and Control, December 2002, Las Vegas, Nevada, USA*, pages 209–215, 2002.

[35] M. Pachter, J. J. D'Azzo, and A. W. Proud. Tight formation flight control. *Journal of Guidance, Control, and Dynamics*, 24:246–254, 2001.

[36] Media Relations. Showcase uav demonstrates 'flapless flight'. Cranfield University Press Release, Corporate Communications, September 27 2010.

[37] A. Richards, J. Bellingham, and M. Tillerson. Coordination and control of multiple uavs. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference (02)*, 2002.

[38] Arthur Richards, Yoshiaki Kuwata, and Jonathan How. Experimental demonstrations of real-time milp control. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2003.

[39] G. Rigatos. Model-based and model-free control of flexible-link robots: a comparison between representative methods. *Applied Mathematical Modelling, Issue 10, October 2009*, 33:3906–3925, 2009.

[40] F. Ruini and A. Cangelosi. Extending the evolutionary robotics approach to flying machines: An application to mav teams. *Neural Networks*, 22:812–821, 2009.

[41] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc., 2003.

[42] M. Salichon. *Learning based methods applied to the MAV control problem*. PhD thesis, Oregon State University, 2009.

[43] M. Salichon and K. Tumer. A neuro-evolutionary approach to micro aerial vehicle control. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1123–1130, 2010.

[44] Jack Shepherd and Kagan Tumer. Robust neuro-control for a micro quadrotor. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 1131–1138, 2010.

[45] T. Shima, S. Rasmussen, A. Sparks, and K. Passino. Multiple task assigments for cooperating uninhabited aerial vehicles using genetic algorithms. *Computations & Operations Research*, 33:3252–3269, 2006.

[46] Dusan Stipanovic, Gkhan Inalhan, Rodney Teo, and Claire Tomlin. Decentralized overlapping control of a formation of unmanned aerial vehicles. *Automatica*, 40(8):1285 – 1296, 2004.

[47] K. Tumer and A. Agogino. Coordinating multi-rover systems: Evaluation functions for dynamic and noisy environments. In *The Genetic and Evolutionary Computation Conference (GECCO 05)*, 2005.

[48] Kagan Tumer and Adrian Agogino. Adaptive management of air traffic flow: A multiagent coordination approach. In *Proceedings of the Twenty Third AAAI Conference on Artificial Intelligence, Nectar Track*, Chicago, IL, July 2008.

[49] George Vachtsevanos, Liang Tang, and Johan Reimann. An intelligent approach to coordinated control of multiple unmanned aerial vehicles. In *Presented at the American Helicopter Society 60th Annual Forum*, 2004.

[50] A. Wu, A. Schultz, and A. Agah. Evolving control for distributed micro air vehicles. In *IEEE Conference on Computational Intelligence in Robotics and Automation*, pages 174–179, 1999.

[51] M. Zabarankin, S. Uryasev, and R. Murphey. Aircraft routing under the risk of detection. *Naval Research Logistics*, 53(8):728–747, 2006.