# ABSTRACT

Title of thesis: AUTONOMOUS SWARMS OF UNMANNED VEHICLES: SOFTWARE CONTROL SYSTEM AND GROUND VEHICLE TESTING

Patricia Jean Kirsch, Master of Science, 2005

Thesis directed by: Professor Bruce L. Jacob
Department of Electrical and Computer Engineering

Unmanned vehicles are being developed for a wide variety of scientific, and military uses. Using a coordinated team allows for greater robustness, flexibility or ease of use. While there are several unmanned vehicle systems in use for a variety of applications, work on swarming has been largely theoretical. This thesis presents Woodstock, a software system implementing a swarming control law. The control law moves the swarm through waypoints specified by GPS coordinates. It was tested using a set of ground vehicles, which are also described. It is shown to be robust to noise, and superior to a previous implementation. The previous implementation was client server, making it vulnerable to a single vehicle loss, while the new system is peer to peer and thus more robust. The system is flexible enough to be portable to other vehicle systems easily.

# AUTONOMOUS SWARMS OF UNMANNED VEHICLES: SOFTWARE CONTROL SYSTEM AND GROUND VEHICLE TESTING

by

Patricia Jean Kirsch

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2005

Advisory Committee:

Professor Bruce L. Jacob, Chair/Advisor
Professor Virgil D. Gligor
Professor Manoj Franklin

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# Chapter 1

# Introduction

This thesis presents the Woodstock system. Woodstock is an autonomous swarming system for meter-scale unmanned vehicles. It is an improvement over other systems in that it is more autonomous and not limited to a single vehicle type. The system presented is based on a previous implementation that was client server. The current version is fully distributed and peer to peer protecting the system from breakdowns caused by the loss of the server vehicle.

## 1.1 Unmanned Vehicles

Unmanned vehicles are becoming popular for reconnaissance and sensing missions, in scientific and military applications. These vehicles can be designed to go where humans cannot. In military applications reconnaissance can also be done without risking human life. Unmanned vehicles include unmanned air vehicles (UAVs), ground vehicles (UGVs), surface vehicles (USVs), and undersea vehicles (UUVs). The vehicles can be autonomous or operator controlled, depending on their application.

Woodstock was tested on a set of UGVs built for this purpose, though the goal is to transition it to other vehicle types, currently the Dragon Eye UAV and the Spartan USV.

## 1.2 Swarming

Swarming or formation control is the use of multiple vehicles to enhance the achievement of objectives. These vehicles move together much as a flock of birds or school of fish do. This can provide more reliability in the form of redundancy. Several small targets are also harder to target than a larger vehicle required to cover a similar area in a scenario such as radar jamming. A leader-follower system can also be used to create unmanned decoys for manned vehicles.

Formation control generally refers to maintaining a set arrangement of the vehicles, while swarming implies a loose organization. Swarming provides only enough organization to get them all to the same place as a group while not colliding. Both can be run autonomously, or in a leader-follower configuration where one or more vehicles follow an operator controlled leader vehicle.

The Woodstock system is currently designed to run as an autonomous swarm, once waypoints are received from a base station. The algorithm is also easily adaptable to running a leader-follower scenario, where the swarm follows an operator controlled leader.

## 1.3 Organization

Chapter two of this thesis presents background information on unmanned vehicles and swarming algorithms. Chapter three presents the control law and the previous version of Woodstock on which this work was based. Chapter four is a description of the software system and the ground vehicle used for testing. Chapter five shows

results from testing of the ground vehicles. Chapter six presents a summary, plans for improving the system and options for transitioning it to various applications.

# Chapter 2

# Background

## 2.1 Unmanned Vehicles

Unmanned vehicles have moved beyond simple R/C vehicles for toys, though a remote control remains the basis of many systems. Vehicles can be equipped with a variety of cameras, sensors, or special equipment to accomplish objectives, though military systems have focused primarily on reconnaissance to this point. The following vehicles are examples of the types of vehicles currently in use or under development. The Woodstock algorithm is targeted at small, meter-scale vehicles such as Dragon Runner, Dragon Eye, or Spartan systems.

### 2.1.1 UAVs

The Navy has used UAVs for monitoring the environment around units for some time [1, 2]. One of the most recent UAVs is the Dragon Eye system, used by the Marines, and developed at Naval Research Lab [3]. It is shown in Figure 2.1. Another is the Firescout system, which is a helicopter-based system still under development [4].

### 2.1.2 UGVs

UGVs are a more recent development. The Dragon Runner system, developed by Naval Research Lab and Carnegie Mellon is designed for urban warfare. It is

Figure 2.1: The Dragon Eye UAV

Figure 2.2: The Dragon Runner UGV

operator controlled, and equipped with a camera and motion sensors. It can be used to scout around corners, as a sentry, and is small and rugged enough to be thrown up stairs or into upper story windows [5]. It is shown in Figure 2.2. Current Naval work is on larger, combat-capable vehicles, such as Gladiator, a modular system capable of reconnaissance, direct fire, and delivery of a variety of payloads [6].

### 2.1.3 USVs

One USV system is the Spartan system from NUWC [7]. The Spartan system is a modular system so it can be deployed for a variety of objectives. Spartan is based on a current manned boat, and is shown in Figure 2.3. NUWC is also working on

Figure 2.3: The Spartan USV

an unmanned sea surface vehicle (USSV) system (USSV is used to be distinctive), which is being designed from the ground up to be unmanned, allowing unique hull designs [8].

## 2.2 Swarming Algorithms

Swarming is a topic of much research, but most of this work has been theoretical [9, 10, 11]. There have been a few projects dealing with physical vehicles. One is using UAVs at West Virginia University. This system uses leader-follower formation control. It deals directly with the aircraft controls, [12]. Another is a UUV project at Princeton, intended for organizing underwater sensors. This system requires the use of a central organizing system to run their algorithm and transmit instructions to the vehicles when they surface [13].

The approach presented by Woodstock is more flexible than either of these, being capable of navigating single vehicles or a swarm to waypoints, needing an outside system only for initializing waypoints. The vehicles also need not be homo-

geneous, as only location information needs to be transmitted, for each vehicle to calculate its own steering information.

## 2.3 Challenges

Implementing a swarming system presents several challenges. These include those inherent in the design of any physical device such as hardware and environment uncertainties. The main ones here are those in determining location, as GPS is not perfect, especially at the scale of the tests performed.

In working with already developed vehicle systems swarming systems also face challenges in the number of software systems that must interact for correct operation. These include whatever controlling or monitoring software the swarming system needs, the actual swarming code, and the already existent and perhaps not well documented vehicle control for steering and speed.

# Chapter 3

# System Specification

The Woodstock system was designed to implement a specific swarming algorithm developed at the University of Maryland. It was based on a previous version of the software, written in Java with a client-server configuration. The new system is written in C++ with a more peer-to-peer approach.

## 3.1 Algorithm Used

The swarming algorithm implemented by the Woodstock system was developed by Dr. Justh and Dr. Krishnaprasad at the University of Maryland. It is described in [14]. It contains rules for four different types of waypoints. They include flythrough, circular, and transitions between these two in both directions. In flythrough, the swarm navigates to within a certain distance of the waypoint before moving on to the next. The position of the swarm is measured by the centroid of all vehicles plus a virtual vehicle at the waypoint. For circular, the swarm circles the waypoint for a specified number of timesteps. There are also transition waypoints in each direction, to help smooth steering, which were implemented but not tested. The waypoint is treated as a virtual vehicle, so it interacts mathematically with the vehicles in the same manner.

The algorithm includes estimation of location at the next timestep. This

means the GPS location information is theoretically only needed to make up for imperfections in the speed control, and obstructions such as hills, winds, or currents. Obstacles are not currently being dealt with, as the main targets for Woodstock are UAVs, and USVs and in water or air obstacles are much rarer.

### 3.1.1 How It Works

The algorithm works by adding together several terms to determine the desired turning of the vehicle. These are then summed across all pairs of vehicles. In the flythrough case, three terms are used, for the circular only the first two. The first aligns the vehicle in being turned perpendicular to the baseline between the two vehicles. The next steers the vehicle towards or away from the second to maintain the desired separation (set as a parameter). The last, used only for flythrough waypoints, aligns the turning vehicle to the other. Each term is weighted by a constant, so the various behaviors can be emphasized.

The algorithm uses several vectors to describe the position and motion of the vehicle, referred to collectively as shape variables. These vectors are $\mathbf{r}_j$ the position of vehicle $j$ ($\mathbf{r}_{jk} = \mathbf{r}_k - \mathbf{r}_j$), $\mathbf{x}_j$, and $\mathbf{y}_j$ are the tangent and normal vectors to the vehicles trajectory.

The full equations for the control laws are in equation 3.1 for flythrough and 3.2 for circular

$$u_j = \frac{1}{n}\sum_{k\neq j}[-\eta(\frac{\mathbf{r}_{jk}}{|\mathbf{r}_{jk}|}\cdot\mathbf{x}_j)(\frac{\mathbf{r}_{jk}}{|\mathbf{r}_{jk}|}\cdot\mathbf{y}_j)+\alpha[1-(\frac{r_0}{|\mathbf{r}_{jk}|})^2](\frac{\mathbf{r}_{jk}}{|\mathbf{r}_{jk|}}\cdot\mathbf{y}_j)+\mu\mathbf{x}_k\cdot\mathbf{y}_j] \quad (3.1)$$

$$u_j = \frac{1}{n}\sum_{k\neq j}[\pm\eta(\frac{\mathbf{r}_{jk}}{|\mathbf{r}_{jk}|}\cdot\mathbf{x}_j)(\frac{\mathbf{r}_{jk}}{|\mathbf{r}_{jk}|}\cdot\mathbf{y}_j)+\alpha[1-(\frac{r_0}{|\mathbf{r}_{jk}|})^2](\frac{\mathbf{r}_{jk}}{|\mathbf{r}_{jk|}}\cdot\mathbf{y}_j) \tag{3.2}$$

These equations are for $n$ vehicles, including the virtual vehicle at the waypoint. The value $u_j$ is the magnitude of the steering change for vehicle $j$ in radians. The values $\eta$, $\mu$, and $\alpha$ are weighting constants, which were all set to 1 for current testing.

The laws for transition waypoints involve a weighted sum of the two for circular and flythrough waypoints. The weighting is set by a smoothing parameter that moves the weighting more towards the one being transitioned to at each time step.

### 3.1.2 Implementation

The first thing that must be dealt with is converting GPS coordinates to an X-Y system. This was done using great circle formulas found at [15]. The first waypoint received is considered the origin of the system, and others are determined by finding the distance and angle from the origin. The angle and distance are then converted from polar coordinates to cartesian. This is done as coordinates are received for an iteration of the algorithm. The overall structure of a iteration of the algorithm is shown in figure 3.1.

For each iteration, the next step is initializing the shape variables. The shape variables are stored in matrices to simplify calculations. The shape variables of the vehicles are all set in reference to the first one. The magnitude of the steering change ($u_j$) is then determined as a running sum based on the waypoint type. The

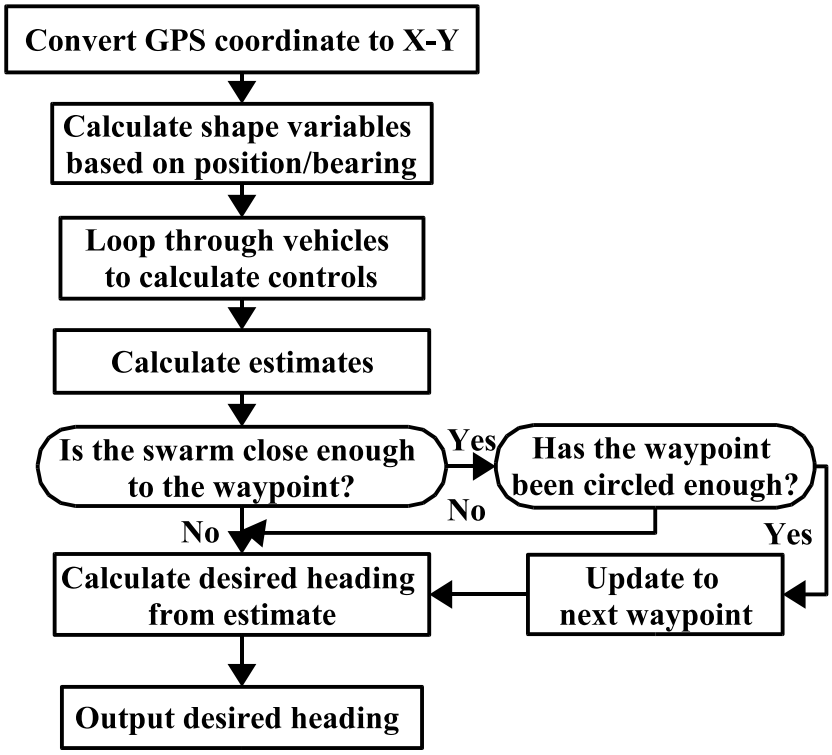


Figure 3.1: Structure of an Algorithm Iteration

magnitude is capped at a parameter $u_{max}$ to help limit steering energy (useful for UAVs).

Once the controls are calculated the state is updated and the new heading found. This first involves calculating the estimates for the next timestep. The vectors are used to calculate the offset from the current position, and the two are added together. The waypoint is then checked, and the management of that is updated if it is decided that the swarm is close enough. The new heading is then found from the bearing of the estimated position.

## 3.2 Previous Version of the System

Much of this work was based on a previous version of the Woodstock system used for feasibility testing of the algorithm. Some of this required reverse engineering as the previous system was not well documented. The hardware used was used with the previous version, though repairs and troubleshooting were required. The mathematical portions of the software were also based on this.

The software system needed to be overhauled to decouple it from a deprecated visualization system. It was decided to port it to C++ from Java as it was more likely that there would be development tools available for other platforms that way. Since many of the decisions were influenced by the original software a description follows.

### 3.2.1 Previous Software Architecture

The software was organized around the Java event model. A listener is registered for the serial port connected to the GPS and to the network, so that events are generated from activity on these ports. This makes the execution somewhat non-deterministic, and the following description is for the relationships between events at steady state.

Each vehicle keeps an array of the current positions of all the vehicles. This is updated by the "master" a single vehicle designated by the base station. When an RMC message (the basic GPS message containing coordinates, speed and course) comes in from the GPS a vehicle will convert this to X-Y coordinates (centered at the first waypoint received by that vehicle), iterate the control algorithm, and send its current location to the master. If the vehicle is the master it also sets a timer to send out the new position vector to all vehicles, and the base station.

The control algorithm uses several constants that are read in from a text file at startup. It assumes knowledge of where all the other vehicles are at each step of the algorithm. Should an update of the position vector not be available the estimates from the previous iteration are used instead. The algorithm is used to calculate headings that are then used to turn the vehicle.

# Chapter 4

# Design

## 4.1 Software

### 4.1.1 Vehicle Software

The software was written in C++ to run under *nix (specifically the Familiar distribution of Linux) but should be easily portable to other systems. Anything that would need extensive updates, such as networking, and serial controls would have to be updated to move to a new system anyway, as their communication and other resources would be different.

#### Communications

Each vehicle maintains its own coordinate system, as a conversion from latitude and longitude, and calculates all controls independently, receiving only raw location/heading updates from the other vehicles. This reduces the amount of information that must be sent, and the system's dependence on it, rather than having to negotiate a coordinate system or get controls from a central source.

All communications are done using UDP packets, a best effort connectionless system. This was chosen for ease of programming, because there is less setup necessary for network communications. The algorithm may perform estimates so

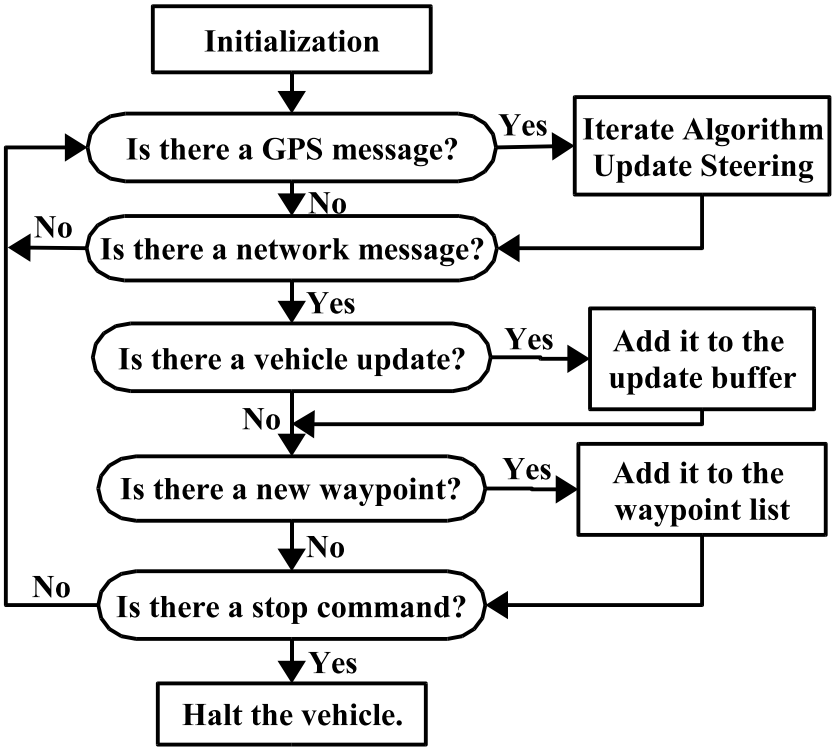


Figure 4.1: Overall Software Structure

the loss of an occasional update would not be disastrous, and initial tests showed good reception with the 802.11b cards to be used.

Overall Structure

The overall structure of the software is shown in Figure 4.1, it starts with initialization of communications and reading in of algorithm parameters from a file. It then enters a loop to receive waypoints from the base station, in the order to be visited. This need not be the final set, more may be added, but only at the end of the set, and the total number pending (the one currently being navigated to and all after) is fixed. This is finished by the base station sending a go command.

The software then enters the main loop of the program. On each run through the main loop the software checks the GPS unit and the network for messages and processes them.

### GPS Messages

On receipt of an RMC message from the GPS the software will update its state. This includes running the algorithm and sending controls to the vehicle. The first step is to check if updates from other vehicles are available. If this is the case then they are passed to the algorithm along with the current position to iterate the algorithm. This returns a new heading for the vehicle. This is then used to adjust the steering of the vehicle by comparing it to the current heading. The vehicles are currently set to run at a constant speed of 1 m/s. If no updates have been received the prior estimates are used, and the steering process is the same.

### Network Messages

A network message received in the main loop will be one of three types, a waypoint, a position update from another vehicle, or a stop message. Waypoint additions are added to the set of waypoints in the algorithm. A position update is added to a buffer of these updates, overwriting any unused one from that vehicle. A stop command causes the vehicle to fall out of the loop and stop.

### 4.1.2 Base Station Software

The base station is used to send the vehicles waypoints, and start and stop them. It is written for Microsoft Windows, reading the waypoint information from a file, then printing GPS information as received from the vehicles. The waypoints and GPS information are currently being expressed in radians. An annotated sample file is shown in Figure 4.2. The 'w' at the beginning of the line indicates this is a waypoint, the id is then given, then the latitude and longitude in radians, then the waypoint type (in order, starting at zero, these are flythrough, circular, flythrough to circular, and circular to flythrough), the number of timesteps this waypoint is to be circled (if circular, otherwise this field is ignored). These lines are sent out to the vehicles verbatim. The waypoints and commands are sent to the vehicles using IP multicast. IP multicast is routing to multiple recipients that is part of the IP layer of a network. Messages sent to specified IP addresses are received by any computer registered for that group (registration is done with the operating system of the computer).

## 4.2 Testing Hardware

The hardware for testing was a set of UGVs adapted from an early prototype of the Dragon Runner system and are shown in Figure 4.3. These vehicles are meant as simple test platforms, and not as a robust unmanned vehicle system. They are built on the chassis of an R/C monster truck, the motor and steering servos are provided

```
225.0.0.37 12345 #IP and port to send out messages
225.0.0.38 4321 #IP and port to receive from vehicles

#waypoints
w1,.6777061,-1.344353,0,0 #w indicates this message is a waypoint
w2,.6777031,-1.344355,0,0 #the integer is the id
w3,.6777017,-1.344356,0,0 #the next two are latitude
w4,.6776998,-1.344356,2,0 #and longitude, in radians
w5,.6776979,-1.344356,1,5 #(N or E positive)
w6,.6776969,-1.344356,3,0 #the last two are type index
w7,.6776955,-1.344356,0,0 #and the number of timesteps to circle
```

Figure 4.2: Annotated Waypoint File Sample

Figure 4.3: Testing Vehicle

by this as well. A hardware quick reference is in Table 4.1 and a block diagram is in Figure 4.4.

The system is controlled by a Intrinsyc Cerfboard single-board computer, using a StrongArm or Xscale processor. It is running the Familiar distribution of Linux. It includes several serial ports used to communicate with the other hardware, and a CompactFlash slot used to hold an 802.11b wireless card. The Cerfboard is shown attached to the panel by which it is mounted to the vehicle in figure 4.5.

The direct control of the speed and steering is done by an HC11 microcon-

| System | Hardware |
|---|---|
| Vehicle | Traxxas Stampede R/C Monster Truck |
| Processing Board | Intrinsyc Cerfboard |
| Microcontroller | Motorola HC11 |
| GPS | Leadtek 9543 |
| Power | Apogee 2P-4S Li-ion pack |

Table 4.1: Hardware Quick Reference

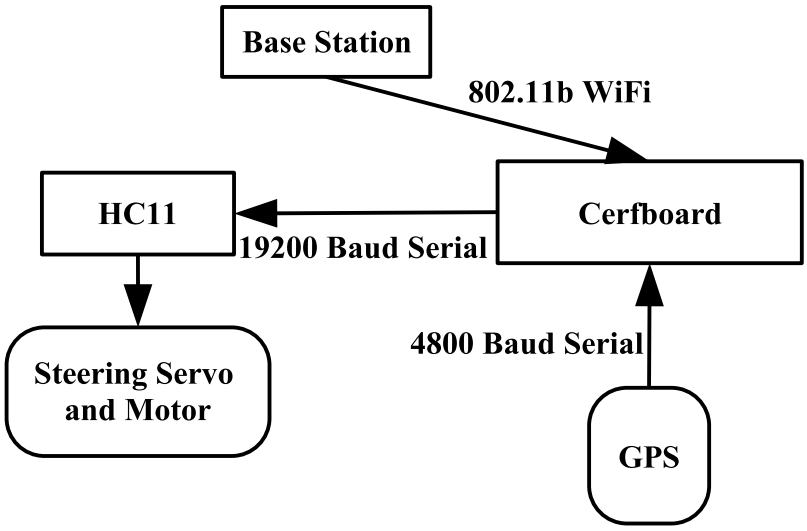


Figure 4.4: Hardware Block Diagram


Figure 4.5: Cerfboard

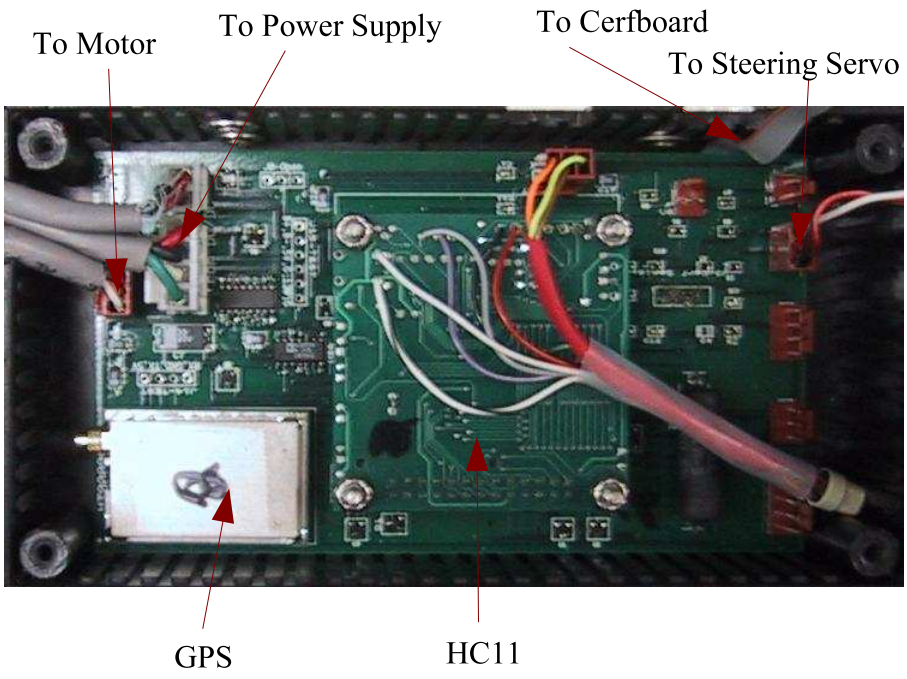


Figure 4.6: The HC11 and GPS on their Interface Board

troller, accepting speed and steering commands through a serial connection at 19200 baud. This then controls the motor and steering servos. The software this board is running is treated as a black box.

Location is measured using a Leadtek 9543 OEM GPS unit. This communicates over serial at 4800 baud. Only the basic RMC message is used, the rest are discarded. The HC11, GPS and their interface board to the rest of the vehicle is shown in figure 4.6.

# Chapter 5

# Testing and Results

## 5.1 Performance Criteria

The system can be judged on two basic criteria: following/arriving at the specified waypoints, and maintaining appropriate intervehicle distances.

The control law that the vehicles run has been shown to meet these criteria in [14]. So the software system must be shown to implement this correctly. Testing was primarily done on a qualitative basis, as the steering and speeds of the vehicles were not exact. This was found in testing of the previous version, where small hills influenced the positions of the vehicles while circling a waypoint, and the speeds could not be adjusted to fix it, as the driving motor gear ratio limited the resolution of the motor speed.

Measurements were also taken to investigate packet losses due to the use of UDP packets for the intervehicle communications.

## 5.2 Test Setup

Testing was done in several parking lots in the area. Most tests were limited to two vehicles due to a limited amount of available space.

The vehicles were set up to maintain a separation of one meter, and the

waypoint radius was set to two meters. The waypoint radius is how close the centroid of the estimated formation must be for the swarm to move on to the next waypoint.

### 5.2.1 Reliability

Reliability was tested based on several failure modes. This included random changes in the measured position, to simulate attacks on GPS. Another mode was a random change in the calculated heading before the steering was changed. These were both run at increments of 10% of actions being affected. The last was randomly losing updates, to simulate a loss of the server in the old implementation. This was used to simulate comparisons of the old system, where if the master (server) were lost communications would be destroyed, to the new system.

#### Steering Changes

Steering changes are meant to simulate changes made in later systems by obstacle avoidance systems or some other need for a vehicle to deviate from its expected path. It also covers interference from wind or waves, though this is somewhat handled by the test areas not being flat. This was done by randomly adding up to $\pm.1$ radians to the heading calculated by the algorithm. Only a single car would make changes, since if all of them were, they would make the same change, and thus not affect the swarm.

GPS Changes

Attacks on the GPS system were simulated by randomly selecting iterations where a random change of up to $\pm 1x10^{-6}$ were added to the radian representation of the position or up to $\pi$ for the course in radians. This limit was chosen to make the change noticeable, but the position still reasonable.

### 5.2.2 Communications

UDP packets were chosen as the main basis for communications as it would be easiest to implement on the already existing 802.11b channel. Since this is a best-effort protocol and transmission is not guaranteed losses are possible. This was tested by making a log of the number of vehicles included in each iteration of the algorithm, any missed vehicles would have been due to communication losses.

## 5.3 Results

### 5.3.1 Correct Navigation

Without any interference from the test system correct navigation was seen in groups of up to 4 vehicles, the maximum amount of hardware available. A sample track for two vehicles is shown in figure 5.1. The sample was limited to two vehicles to make the paths clearer. This was generated from the X-Y coordinates of the vehicles positions.

While collisions did occasionally happen, they were limited to the beginning

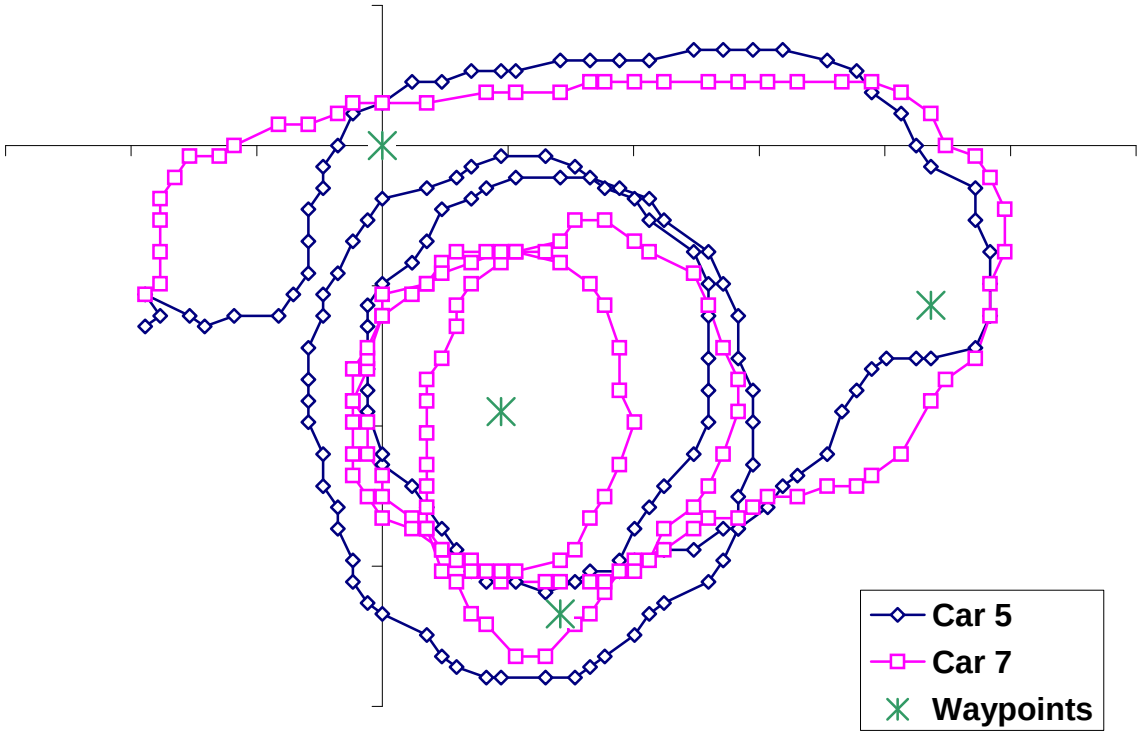


Figure 5.1: A Sample Track For 2 Vehicles

of a run. This is likely due to GPS initialization, especially for heading, taking some time, rather than a problem with the system in general.

### Comparison to Old System

While the old system had similar performance in a normal test, it is much more susceptible to failure, due to the client-server configuration. A simulation of server loss was done by dropping the updates after the system had stabilized, forcing the system to run on estimates. While each individual car was capable of navigating to waypoints on its own, swarming behavior broke down without any additional interference from the test system. Collisions were commonly observed, at any point in

| % Failure | Behavior Changes |
|---|---|
| 10 | Changes evident |
| 20 | Intermittent separations |
| 30 | Areas covered multiple times |
| 40 | |
| 50 | |
| 60 | |
| 70 | Several collisions, large separations |
| 80 | No organization evident, only 2 waypoints reached |

Table 5.1: Results for Steering Changes

the run. Other anomalous behavior included vehicles heading in opposite directions, and separating by large distances.

This indicates that the old system is highly vulnerable to one well placed vehicle loss. In most situations the loss of a leader can be dealt with through dynamic reassignment. This is not feasible in the case of the Woodstock system since the vehicles are unlikely to have independent failure modes. There is no way to know that the new selction will not be the next vehicle shot down. The new system, being peer to peer, keeps the loss of any single vehicle from affecting the whole group.

Steering Changes

Changes caused by the steering "failures" are summarized in table 5.1. Behaviors continued as the failure rate was increased until superseded by something else. Circling was generally less affected than finding waypoints. The system was able to recover from a failure rate of up to 70%. These rates may be inflated by the small testing area, allowing more chance successes.

| % Failure | Behavior Changes |
|---|---|
| 10 | |
| 20 | |
| 30 | Areas covered multiple times |
| 40 | |
| 50 | |
| 60 | Some Collisions |
| 70 | Several collisions |
| 80 | No organization, could not be kept within testing area |

Table 5.2: Results for GPS Changes

GPS Changes

Behavior under GPS changes was similar to that with steering changes. This is summarized in table 5.2. The system could recover from a failure rate of up to 70%.

Accurate GPS is needed to the required resolution of the waypoints. It was commonly observed during testing that the GPS system on the vehicles and that used for setting waypoints did not agree, with the vehicles making circles about a point several feet from where the waypoint was taken.

### 5.3.2 Communications

To investigate how much this influences the system performance was measured and are shown in Table 5.3. This was for a test with three vehicles. The number of updates received at each time step was recorded, and the count for each number for each car is listed in the table.

All updates were received approximately 75% of the time. Even with these losses the system could still perform as required. This shows that both low-overhead

| | 3 updates | 2 updates | 1 update | Total |
|---|---|---|---|---|
| Car 4 | 99 | 32 | 13 | 144 |
| Car 5 | 99 | 34 | 11 | 144 |
| Car 6 | 129 | 13 | 2 | 144 |
| Totals | 327 | 79 | 26 | 432 |

Table 5.3: Performance of the Communication System

communication systems are appropriate for the Woodstock system, but that it is quite robust to intermittent communications losses.

# Chapter 6

# Conclusions and Further Work

## 6.1 Summary

Swarming teams of unmanned vehicles provide extra resiliency and flexibility to achieve sensing or reconnaissance missions. Woodstock, a software system implementing an algorithm for control of such systems was presented. It is more reliable than the previous version due to its peer to peer communication architecture. The system is robust enough to deal with high levels of noise in location and control. It is flexible enough to be easily portable to other platforms and vehicle types.

## 6.2 Further Work

Further work on the swarming vehicles includes increasing the robustness of the software system, and transitioning it to other applications. The base station software would also have to be more user friendly before it could used for anything but testing.

### 6.2.1 Increasing Robustness

The main area for improving the robustness of the software is in the network communications. This could be increased by handshaking or another protocol to ensure transmissions go through.

Another point that may cause problems is a single vehicle of the swarm

becoming physically stuck somewhere. Since the swarm determines when to move on to the next waypoint based on the location of all vehicles, this could result in the swarm not moving on, and becoming stuck at the current waypoint.

Both of these problems point to the need for a more detailed handling of vehicles joining and leaving the swarm. This would allow an individual update loss to be estimated, rather than just when none are received, since whether the vehicle has left or not is known. It would also allow dropping of a vehicle that does not seem to be making progress.

### 6.2.2 Applications

The initial objective for transitioning the swarming software is the Dragon Eye UAV, starting with a hardware in the loop simulation. Talks are currently underway with NUWC to integrate it with either their Roboski or Spartan USV. Both cases will require a new controller system. GPS and communications will have to be re-evaluated based on available resources as well. In the case of the UAVs the control algorithm will have to be updated to deal with three dimensions, and those changes integrated into the software. The algorithm work is currently underway at the University of Maryland. The current plan for the USVs is a leader-follower configuration, rather than following waypoints. This would require removing the waypoint portion of the algorithm code, and implementing a speed controller, so the speed of the leader could be matched.

# BIBLIOGRAPHY

[1] M. Peck, "The UAV That Wouldn't Die." [Online], Oct. 2002. http://www.military-aerospace-technology.com/article.cfm?DocID=293.

[2] D. Brown, "BAMS,Eagle Eyes, and Dragon Eyes." [Online], 2003. http://www.navyleague.org/sea_power/apr_03_66.php.

[3] "Dragon Eye Unmanned Aerial Vehicle." [Online]. http://www.jfcom.mil/about/experiments/mc02/concepts/drageye.htm.

[4] "Fire Scout UAV Continues Flight Tests." [Online], 2002. http://www.news.navy.mil/search/display.asp?story_id=2105.

[5] P. Thompson, "Warfighting Lab tests unmanned ground vehicle." [Online], 2002. http://www.marines.mil/marinelink/mcn2000.nsf/lookupstoryref/200242613519.

[6] C. Walton, "Future and present meet in unmanned ground vehicles." [Online], Dec. 2003. http://www.usmc.mil/marinelink/mcn2000.nsf/0/1948D6915CF48BFE85256DF10074E88E?opendocument.

[7] "Spartan Deployed on Gettysburg." [Online], Dec. 2003. http://newshome.news.navy.mil/search/display.asp?story_id=10964.

[8] W. Palmer and R. Brizzolara, "West Bethesda Helps Implement Unmanned Vehicle Technology." [Online], Feb. 2004. http://www.dt.navy.mil/wavelengths/archives/2004_02.html.

[9] A. Jadbabaie, J. Lin, and A. S. Morse, "Coordination of groups of mobile autonomous agents using nearest neighbor rules," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, 2003.

[10] P. Ogren, E. Fiorelli, and N. Leonard, "Formations with a Mission: Stable Coordination of Vehicle Group Maneuvers," in *Proc. of the 15 International Symposium on Mathematical Theory of Networks and Systems,*, 2002.

[11] D. Chang, S. Shadden, J. Marsden, and R. Olfati-Saber, "Collision Avoidance for Multiple Agent Systems," in *Proc. 42nd IEEE Conference on Decision and Control*, 2003.

[12] B. Seanor, G. Campa, G. Yu, M. R. Napolitano, L. Rowe, and M. G. Perinschi, "Formation Flight Test Results for UAV Research Aircraft Models," in *Proc. of the 2004 AIAA Intelligent Systems Technology Conference*, 2004.

[13] E. Fiorelli, N. E. Leonard, P. Bhatta, D. Paley, R. Bachmayer, and D. M. Fratantoni, "Multi-AUV Control and Adaptive Sampling in Monterey Bay," in *Proc. IEEE Autonomous Underwater Vehicles 2004: Workshop on Multiple AUV Operations (AUV04)*, 2004.

[14] E. Justh and P. Krishnaprasad, "A Simple Control Law for UAV Formation Flying," tech. rep., Institute for Systems Research, 2002.

[15] E. Williams, "Aviation Formulary V1.42." [Online]. `http://williams.best.vwh.net/avform.htm`.