Rapid Verification and Validation of Strategic Autonomous Algorithms Using Human-in-the-Loop Architectures

Christopher W. Lum*, Matthew L. Rowland†, Rolf T. Rysdyk‡, and Juris Vagners§

Autonomous Flight Systems Laboratory

University of Washington, Seattle, WA, 98195, USA

Much unmanned aerial vehicle research aims to minimize the need for human interaction. This is often achieved by developing algorithms which allow a single agent or possibly a team of agents to operate autonomously. Most of these mission management algorithms operate at a high, strategic level and assume that low level tasks such as vehicle state stabilization and payload signal tracking have already been realized. The difficulty in verifying and validating strategic algorithms in flight tests is that implementing these algorithms requires the development of many other lower-level subsystems which are not directly related to the strategic algorithms. This paper presents an architecture and hardware implementation of a ground based, distributed testing environment that can be used to test strategic level algorithms in an efficient manner. This system allows human interaction at very specific points to avoid developing a fully autonomous system, but still preserves the function and contributions of the strategic algorithm. This architecture and ground based testing facility greatly reduces development time and allows algorithms to be tested with few approximations before implementing them on a fully autonomous system. Human subject results in ground simulation and flight testing are presented to verify the approach.

Nomenclature

$f_0()$	Instantaneous cost function for path following problem
$J()$	Total cost for path following performance
P_E, P_N	East and north position of agent, respectively
p_A	Starting waypoint of current track segment
p_B	Ending or active waypoint of current track segment
p_i	Waypoint i of path
r_{max}	Distance agent can travel in a single step
t_i	Desired arrival time at waypoint p_i
x_0	Agent's starting coordinate of path
x_d	Agent's desired ending coordinate of path
$x_{agt}(t)$	Agent's location at time t ($\in \Re^2$ or $\Re^3$)
$x_p(t)$	Point on path segment closest to $x_{agt}(t)$
ΔT	Time between steps/waypoints
$\kappa^\star$	Scalar parameter used when computing pilot path following performance
ψ	Heading

*Research Assistant, Dept. of Aeronautics and Astronautics, lum@u.washington.edu, AIAA student member

†Instructor, Dept. of Civil and Mechanical Engineering, United States Military Academy, matthew.rowland@us.army.mil

‡Advanced Development, Insitu, rolf.rysdyk@insitu.com, AIAA member

§Professor Emeritus, Dept. of Aeronautics and Astronautics, vagners@aa.washington.edu, AIAA member

I. Introduction

Many modern research projects in the field of unmanned aerial vehicles (UAVs) are directed toward the development of algorithms which govern the actions of a single UAV or a team of UAVs at an abstract level. An autonomous system may refer to any system which contains decision making capabilities on some level. These systems typically start with a vehicle such as an aircraft or boat. These vehicles are augmented with autonomous algorithms to modify their behavior and increase their capabilities. A vehicle augmented in this manner with basic decision making capabilities is referred to as an agent. An autonomous agent is a complex system comprised of many separate algorithms.

Often, the terms "autonomous systems" and "autonomous algorithms" are used interchangeably. The distinction made in this work is that autonomous algorithms are the routines which manage specific tasks without human interactions and autonomous systems are comprised of many sub-systems including hardware and autonomous algorithms. The autonomous system may refer to a single actual agent or the entire team and infrastructure to manage a group of agents.

It is useful to define a metric for determining the class of a given autonomous algorithm. One common metric is to classify an autonomous algorithm by the level of autonomy it achieves and types of tasks it handles. Three common classifications are shown in Figure 1.

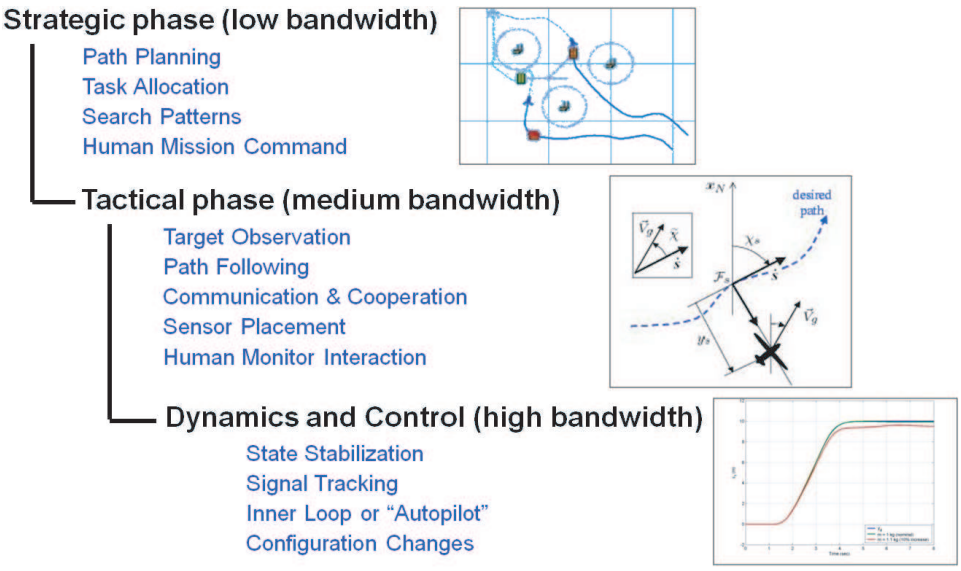


Figure 1. Different levels of autonomy.

Figure 1 shows the first level of autonomy as the strategic level. These algorithms often operate at a low bandwidth and may be in charge of tasks such as mission planning. They are sometimes referred to as "high level control schemes." Next, the tactical level of control operates underneath the strategic level. This level of control is concerned with more specialized tasks such as path following or orbit coordination. Lastly, the dynamics and control level consists of the classical control problems such as state stabilization or signal tracking. These are the most specialized and "narrow-minded" and operate at a high bandwidth.

Note that solutions of problems at the dynamics and control level are considered a fairly mature technology with limited research directed towards new innovations. Controllers operating at the dynamics and control level only offer rudimentary autonomous capabilities to the agents. In terms of increasing the autonomy of a system, much more can be gained by developing algorithms at the tactical and strategic levels. Despite this fact, a viable autonomous system must use algorithms from all different levels to work together to achieve a common goal.

This hierarchical structure of automation and the challenges it entails has been studied and analyzed by groups such as Passino et al.[1] Others such as Clough[2] , Huang et al[3] , and Sholes[4] have proposed more detailed metrics for measuring a system's level of automation that operate in a similar fashion but provide a more detailed classification system.

Many strategic algorithms govern high level behaviors such as task and path planning, but assume that

low level algorithms already exist to handle tasks such as state stabilization and signal tracking. This creates complications at the verification and validation stage because a fully autonomous flight test requires significant logistical planning and development. Many other UAV subsystems that are not directly related to the core strategic algorithm must be developed in order to support the mission. Furthermore, these autonomous flight tests must be conducted in controlled airspaces and under strict supervision. All of these factors greatly increase complexity and development time of the overall system. Many of these problems can be addressed simply by introducing human decision making and interaction at very specific points in the system.

This paper presents an architecture and development of a ground based, distributed testing environment which may be used to test high level algorithms by using a human operator in place of several low level systems. In this fashion, the overall system operates in a manner very similar to the fully autonomous system. This approach offers many benefits. The main advantage is that these high level algorithms can be implemented and tested much faster and with significantly less effort. In addition, applications can be developed for a standard Windows based environment instead of embedded real time systems. Furthermore, these algorithms can be tested in unrestricted airspace due to the fact that they are operating as a pilot-assistance system which the pilot is free to ignore at any point.

The main drawback with this architecture is the introduction of a human operator or pilot who may behave in a non-deterministic fashion. To alleviate this problem, the distributed ground based simulator is used to train potential pilots to interact efficiently and consistently with the system.

Many other research laboratories use ground based simulators to verify and validate control algorithms. The UAV Lab at Georgia Tech has developed significant software and hardware[5] for testing algorithms and implementing flight tests.[6] Many of the interfaces and architecture presented in this paper are based on similar ideas. Previous work at the University of Washington regarding simulation[7,8] and flight testing[9] of autonomous algorithms has led to the development of the current ground based testing architecture.

This paper details the control architecture used to integrate human operators with autonomous algorithms in order to perform verification and validation of high level control algorithms in an efficient manner. The paper also discusses the distributed ground based simulator that is used to simulate actual flight conditions and train human pilots. An example of a pilot interacting with a path planning algorithm[10] is used to illustrate these ideas. Section II introduces the different system architectures which are used in both the ground testing environment and also for flight tests. The various hardware and software components used for supporting both the simulator and flight tests are described in Section III. Interfacing and training of human operators is discussed in Section IV. Results from a successful flight test for a selected mission are shown in Section V. In Section VI, some conclusions and lessons learned are given.

II. System Architecture

A large amount of research and interest lies in developing strategic autonomous algorithms and govern high level behaviors such as task and path planning. Often, it is assumed that that low level algorithms already exist to handle tasks such as state stabilization and signal tracking. This section describes a ground based, distributed testing environment which can be used to test high level algorithms by using a human operator in place of several low level systems. In this fashion, the overall system operates in a manner very similar to the fully autonomous system.

A. Autonomy Architecture

For a fully autonomous system to operate successfully, control laws must be designed and implemented to address all levels of autonomy. An example of a typical setup for a fully autonomous system is shown in Figure 2.

In this model, external inputs such as disturbances are not considered. Low and middle level control tasks are handled by an inner loop controller and the strategic algorithms are responsible for more abstract, higher level mission management tasks. This type of architecture works well to partition the workload and assign tasks to the controllers. Both the inner and outer loop control laws may be implemented using onboard microprocessors or embedded controllers.[11]

Although this architecture is efficient, the main drawback is the fact that although the inner loop control laws can be designed and tested independently from the outer loop controller, the reverse is not true. A

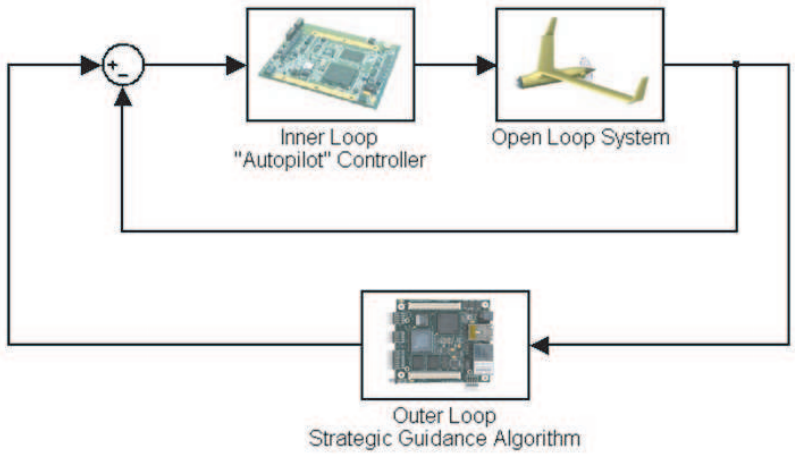


Figure 2. Desired system architecture for fully autonomous flight.

valid inner loop must be implemented in order to verify and validate the outer loop. Furthermore, the inner loop is vehicle specific and must be designed and implemented individually for each vehicle. To further complicate matters, often these inner loop applications must be developed to run on embedded controllers, thus requiring specialized training and development environments.

B. Human-in-the-Loop Architecture

Low level control problems such as state stabilization and signal tracking are considered to be mature technologies with limited academic research currently focused on these problems. However, verification and validation of strategic and tactical control algorithms is currently an active field of research.[12] The general algorithm architecture to support a flight test shown previously in Figure 2 requires excessive overhead for initial verification and validation of strategic algorithms, so approximations are made to support rapid prototyping.

1. Flight Test Architecture

The architecture used in this work for semi-autonomous flight tests that replaces the fully autonomous system with a human pilot and flight vehicle is shown in Figure 3.

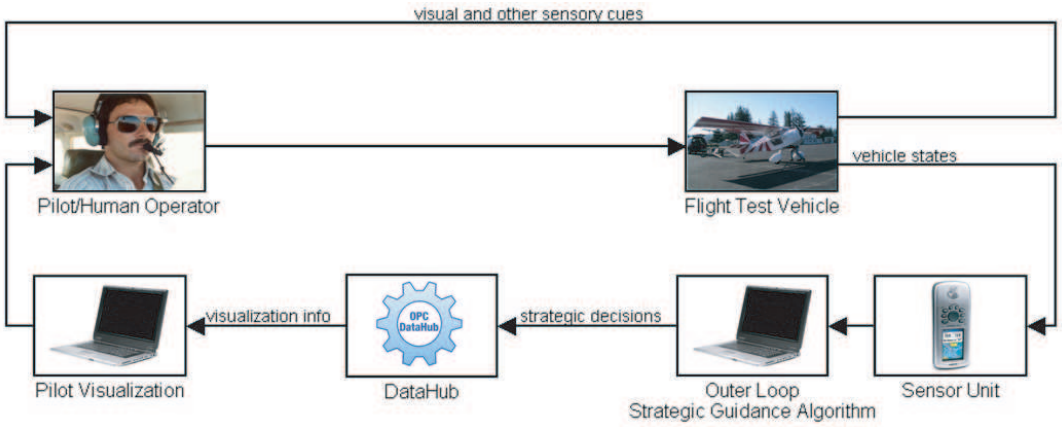


Figure 3. System architecture for human-in-the-loop flight test of strategic controller.

In this setup, the strategic control laws are implemented on a standard laptop PC (see Section III.B).

The tasks that are usually handled by the inner loop controller are instead managed by the human pilot. The outer loop relays information to the inner loop (pilot) by displaying pertinent information to a second laptop (called the visualization laptop). The visualization and strategic laptops are connected via software called OPC DataHub (see Section III.A).

There are several advantages to this architecture. The most obvious is that this setup avoids the significant time and effort required to develop a viable inner loop controller for the vehicle. Instead, the human pilot operates as the inner loop controller by taking commands from the outer loop (strategic guidance algorithms). This is similar to the difference between driving a car assisted by GPS navigation and developing a fully autonomous car capable of navigating via GPS.

This setup allows the strategic algorithms to be developed using a standard development environment such as Microsoft Visual Studio or Matlab since the application may be implemented on a laptop PC running a Windows operating system. This further saves time and effort because it is not necessary to port algorithms to an embedded real time operating system (RTOS).

In addition, this architecture allows vehicles using these strategic algorithms to operate in normal airspace with little or no safety problems (the pilot can choose to ignore commands from the strategic algorithm at any time).

2. *Distributed Simulation Architecture*

The architecture described in Figure 3 is suitable for flight testing with an actual vehicle. An equivalent architecture suitable for simulation and software or hardware-in-the-loop verification is shown in Figure 4.

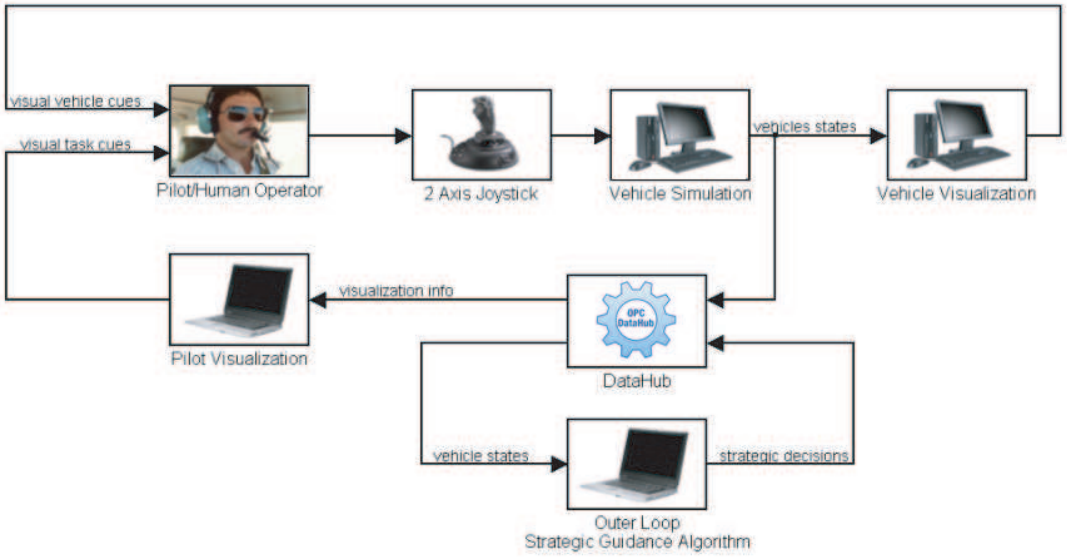


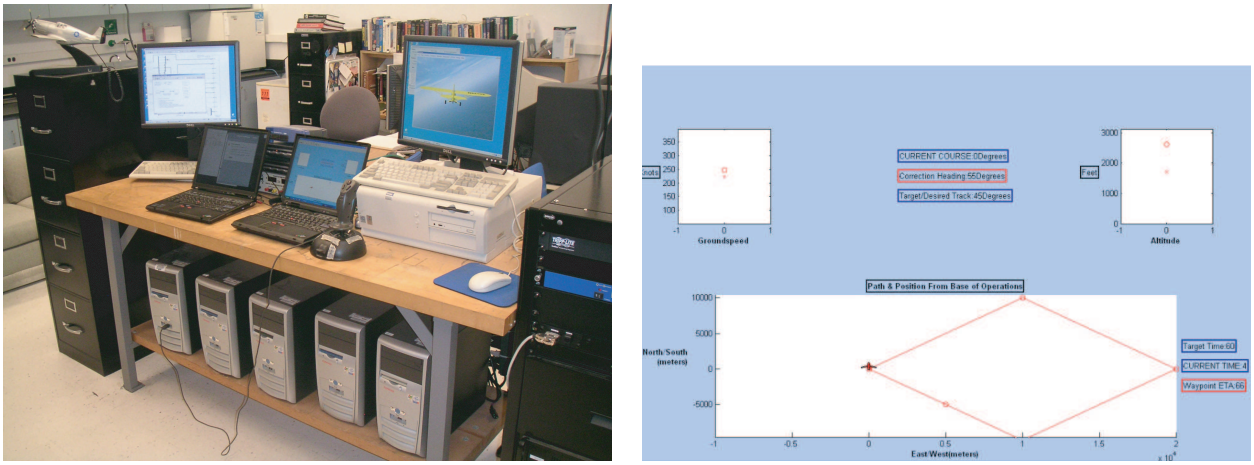
Figure 4. System architecture for ground based distributed human-in-the-loop simulation.

This setup mirrors the flight test architecture shown previously in Figure 3 except the flight test vehicle and sensor unit are replaced with their simulated counterparts. The vehicle is modeled as a 6 degree of freedom rigid body based on the Research Civil Aircraft Model.[13] The aircraft simulation is implemented on two separate desktop PCs and is described in detail later in Section III.

Besides ease of use and low overhead to run tests, this serves the purpose of training operators to interface with the system. In order to accurately validate the outer loop controller, the inner loop controller must behave in a reliable, deterministic fashion. In other words, the human operator must interface with the system in a predictable manner. The amount of error introduced by the human operator depends on the amount and type of practice and training that they receive. The system architecture in Figure 4 can be used to train operators in a variety of scenarios. Items such as the plant model, the strategic tasks, and other factors can be easily modified using this architecture.

III. Hardware and Software

The human-in-the-loop distributed simulator is comprised of several software and hardware systems running in parallel. An example of this system operating during a search mission[14] is shown below in Figure 5.



(a) Physical setup of distributed simulator

(b) Pilot visualization for path following task

Figure 5. Physical setup of Distributed Human-in-the-Loop Simulator and screen shot of operator visualization.

The physical setup of the distributed human-in-the-loop simulator is shown in Figure 5. In this system, the pilot interacts with the simulator via a three axis joystick and receives visual vehicle cues regarding the state of the aircraft from the FlightGear[15] output.

Information regarding the output of the strategic algorithm is displayed to the pilot via a dedicated laptop. A screenshot of the information conveyed to the pilot is shown in Figure 5(b). In this situation, the strategic algorithm determines the path for the agent to follow and displays the pertinent information (visual task cues) to the human operator through the visualization laptop. In the example shown, the path is consists of a diamond shaped flight path that the pilot is required to fly.

The output consists of four separate screens. In the upper left, a display showing the desired groundspeed and the current groundspeed is displayed. In the upper right, a similar display is used for altitude tracking. The display on the bottom consists of a top view of the current path and the location and orientation of the agent. Directly above this wire frame drawing is a series of numbers indicating the current course angle, the desired course angle, and a correction course angle which will put the agent back on track if there is a non-zero cross track error.[16]

A. Software

Software applications used in the distributed simulator are all implemented on Windows XP machines. The vehicle simulation is performed using Matlab Version 7.2.0.232 (R2006a) and Simulink 6. In addition, the AeroSim blockset version 1.2[17] is used to interface to the two axis joystick with the Simulink model.

To distribute processing power, the vehicle visualization is handled by a separate desktop PC. This PC renders the vehicle and environment in 3D using FlightGear v0.9.8. Once again, the AeroSim blockset is used to interface the Simulink model of the vehicle with this visualization tool.

Another crucial piece of software is the OPC DataHub. This is a centralized database with a publish/subscribe architecture where multiple applications can both write to and read from a centralized server. This software is used to transfer data between different applications. Many applications have tools for interfacing with the DataHub. The OPC Toolbox in Simulink allows signals from the Simulink model to be written to (or read from) the DataHub. The pilot visualization is also implemented using Matlab and Simulink and uses the OPC Toolbox to read relevant information from the DataHub.

The strategic algorithm is implemented as a stand alone executable. The flexibility in this approach is that the strategic algorithm can be generated using whatever development environment is most convenient. For example, the strategic algorithm described later in Section IV is developed as a C++ application developed

in Microsoft Visual Studio. Various libraries allow the application to interface with the DataHub and publish relevant data to the database. This allows separate software objects to be dedicated to interfacing with the sensor and the DataHub. Software applications used in the system are designed to be modular so that they can be developed and improved independently.

B. Hardware

The distributed simulator is made up of several hardware components as well. The main hardware is the series of networked desktop and laptop PCs. The various computers, the applications they host, and their functions are summarized in Table 1.

Table 1. Hardware and software components used by various machines in distributed simulator.

Type	Applications	Function
Desktop PC	Matlab Version 7.2.0.232 (R2006a), Simulink 6 with AeroSim version 1.2 blockset and OPC Toolbox, OPC DataHub version 6.3.14.166	Vehicle state and environment simulation and human interface system
Desktop PC	FlightGear v0.9.8	Vehicle visualization
Laptop PC	Matlab Version 7.2.0.232 (R2006a), Simulink 6 with OPC Toolbox, OPC DataHub version 6.1.9.133	Pilot Visualization and OPC DataHub server
Laptop PC	Strategic Algorithm	Strategic Algorithm carrier and sensor interface

The Desktop PC responsible for the vehicle simulation is also responsible for interfacing with the human operator. This is done via a three axis Microsoft Sidewinder joystick.

For flight tests, the vehicle and sensor model are replaced with an actual aircraft and sensor. These are described in Section V.A.

IV. Human-in-the-Loop Interface

Human pilots can be trained on this simulator to interact with the autonomous algorithms. The performance of the operators is also analyzed in this section using various scenarios and metrics.

A. Human Performance Metrics

Although the architecture in Figure 4 may be used to verify and validate a generic, strategic algorithm, several portions must be made specific to the desired task at hand. In this example, the strategic algorithm is a searching algorithm[14] which coordinates a team of agents to search for a target in a two dimensional domain. This algorithm performs many complex calculations which the inner loop (human operator) does not need to know about. From the agent's perspective, the strategic algorithm will simply specify paths for it to follow, it does not require information about why the paths were generated. The distributed simulator is used to train pilots to perform a path following task.

A feasible path for the agent consists of a sequence of waypoints ($x_i \in \Re^3$) where each consecutive waypoint is no more than a distance r_{max} away from the previous one. In addition, each of these waypoints specifies a time, t_i when the agent must arrive at the location x_i. Once the path is determined and displayed to the pilot, it now becomes the job of the pilot to follow this path to the best of their ability.

The skill and ability of the pilot is measured using a performance metric. The position of the simulated agent is recorded every Δt seconds and is denoted $x_{agt}(t)$. At each time t, the agent should ideally be located on the line segment joining waypoints x_i and x_{i+1} for some $i \in \{0, 1, \ldots, d-1\}$ (for the case of $i = 0$, $x_0 = x_{agt}(0)$). For convenience, the point x_i is referred to as x_A (the previous waypoint) and the point x_{i+1} is referred to as x_B (the next waypoint). x_A and x_B have respective time stamps t_i and t_{i+1}. If the agent is not on this line segment, the agent is off the path and the point $x_p(t)$ can be found which is the point on the line segment with minimum Euclidean distance from the agent's current location, $x_{agt}(t)$. The point $x_p(t)$

at each time step is given by

$$x_p(t) = x_A + \kappa^\star(x_B - x_A) \tag{1}$$

In this situation, $\kappa^\star$ is a scalar in the range of $[0, 1]$ which denotes how far from x_A to x_B the point $x_p(t)$ is. It is obtained by solving a minimization problem of $\kappa^\star \in \arg\min\ f_0(\kappa, t) = \frac{1}{2}||x_{agt}(t) - [x_A + \kappa(x_B - x_A)]||^2$ over all $\kappa \in [0, 1]$. The solution can be analytically found to be

$$\kappa^\star = \begin{cases} 1 & \text{if } \varphi > 1 \\ 0 & \text{if } \varphi < 0 \\ \varphi & \text{otherwise} \end{cases} \tag{2}$$

$$\text{where } \varphi = \frac{x_A^T(x_A - x_B) + (x_B^T - x_A^T)}{x_A^T x_A - 2x_A^T x_B + x_B^T x_B} x_{agt}(t)$$

So the instantaneous cost at time step t is given by

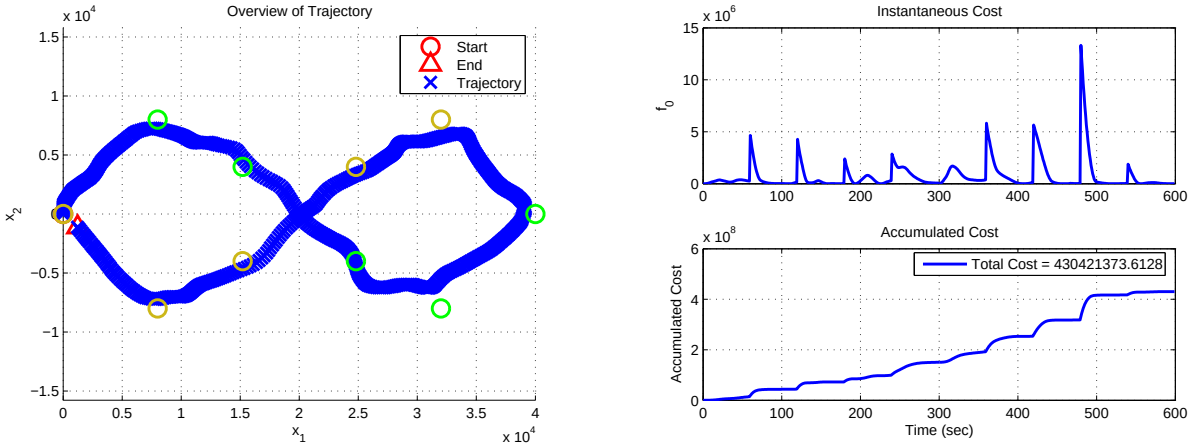
$$f_0(\kappa^\star, t) = \frac{1}{2}||x_{agt}(t) - [x_A + \kappa^\star(x_B - x_A)]||^2 \tag{3}$$

The accumulated cost up to time t is simply the sum of all the instantaneous costs up to the current time

$$J(t) = \sum_{\tau=0}^{\tau=t} f_0(\kappa^\star, \tau) \tag{4}$$

B. Path Following Example

An example of an unskilled pilot flying through two paths and the instantaneous and accumulated cost is shown in Figure 6.



(a) Overview of trajectory

(b) Instantaneous and accumulated cost

Figure 6. Simulator results from human-in-the-loop simulation.

In this situation, the strategic algorithm determines the two paths shown as green and gold circles. This information is then related to the pilot through visual task cues on the visualization laptop. These cues consist of the wire frame representation of the paths and also other information as shown previously in Figure 5(b). The pilot is then tasked with flying the specified figure eight pattern. The pattern is broken up into two distinct paths. The first path consists of five green waypoints (p_i for $i = 1, ..., 5$), each of which have desired arrival times of t = 60, 120, 180, 240, and 300 seconds, respectively. At t = 300, a second path is generated which consists of the five brown waypoints which return the agent to the starting point.

The pilot's performance is measured and displayed in Figure 6(b). The instantaneous cost is a measure of how far off the desired path the pilot is. Notice that the discontinuities in instantaneous cost occur when the active waypoint changes from x_i to x_{i+1} (for path 1, this occurs at t =60, 120, 180, 240, and 300 seconds).

These jumps are due to the fact that at time $t < t_i$, the agent has not reached waypoint x_i but is roughly on the desired path between x_{i-1} and x_i. This results in a low cost (most likely that $\kappa^\star \in [0, 1]$). However at $t > t_i$, the next active waypoint becomes x_{i+1} and since the agent has not yet reached point x_i, the cost becomes large (most likely that $\kappa^\star = 0$).

The performance of the pilot can be judged by the instantaneous cost trace. Skilled pilots will have a low average value with minimal discontinuities. For identical paths and times, the performance can also be judged by $J(t_f)$ which provides a type of score for the run.

The results of a human pilot flying 4 runs over a single path are shown in Figure 7. As can be seen in this figure, the human operates as a nonlinear, adaptive, quickly learning inner loop controller and is able to obtain very good results within just 4 training runs with the system.

V. Flight Testing

The last step in the verification and validation process of the autonomous algorithms involves a real time flight test. This section describes the various components necessary for the flight test. Section A describes the hardware used during the setup and how it integrates with the overall system architecture described previously in Figure 3. The setup and experimental methodology of the test are described in Section B. The results and analysis of the successful flight test are detailed in Section C.

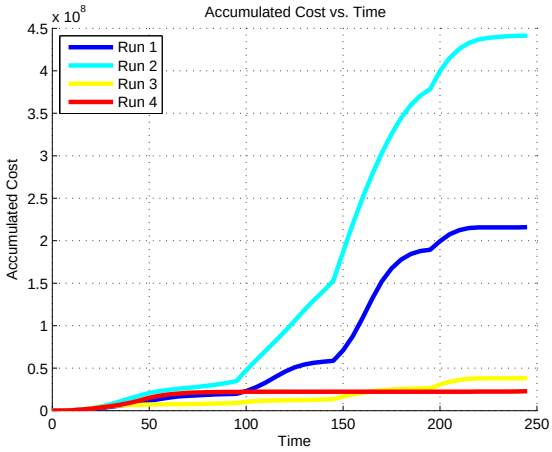


Figure 7. Improvement of human operator performance over 4 runs.

A. Hardware

The main hardware for the flight test involves equipment used to support the autonomous algorithm and the flight test vehicle itself.

1. *Algorithm Hardware*

Two laptop computers make up the main hardware components of the flight test. These machines and their functions were described previously in Section III.B. One computer is the strategic algorithm computer and is responsible for computing the high level, mission management algorithms for each agent. It is also responsible for interfacing with the GPS device and publishing relevant data to the DataHub. The second computer is called the visualization computer and responsible for maintaining the DataHub server and displaying the pertinent information to the human operator.

One major difference between the distributed simulator and the flight test is that the vehicle states are not simulated. For example, during the flight test, the position of the aircraft is measured using a Garmin GPSMap76CSx device connected to the Strategic Algorithm laptop via a serial cable. This GPS device outputs NMEA sentences over a standard RS-232 port at a rate of 1 Hz. The simulation application actively polls the COM port and interprets GPGLL sentences in real time to obtain agent position. The latitude and longitude encoded in the GPGLL sentence is translated into the North and East positions from the specified set of coordinates (the base) using the Vincenty Formula[18] using the WGS-84 Earth model.[19] GRMZ sentences are interpreted to obtain the agent's altitude in real time. The serial port is configured for a 9600 Baud rate. The various hardware and their respective connections are shown in Figure 8.

The components are enclosed in a custom made case designed to isolate the hardware components from the aircraft vibrations (typically 5500-6000 RPM at 65 MPH). The hardware carried in the vehicle during flight tests is shown in Figure 9.

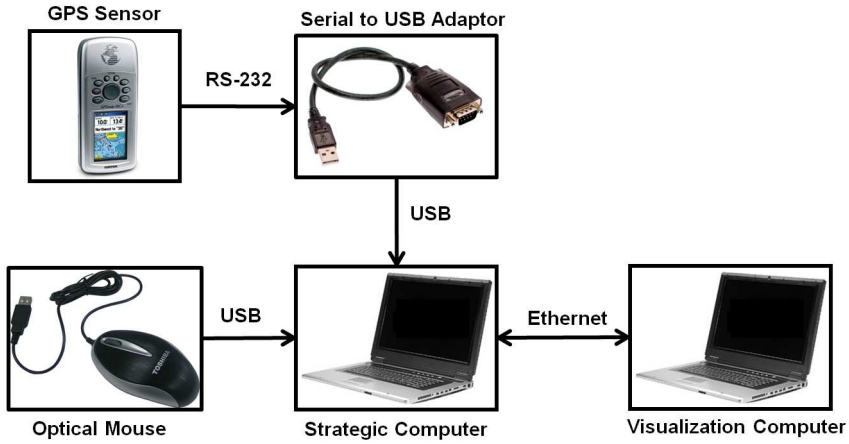
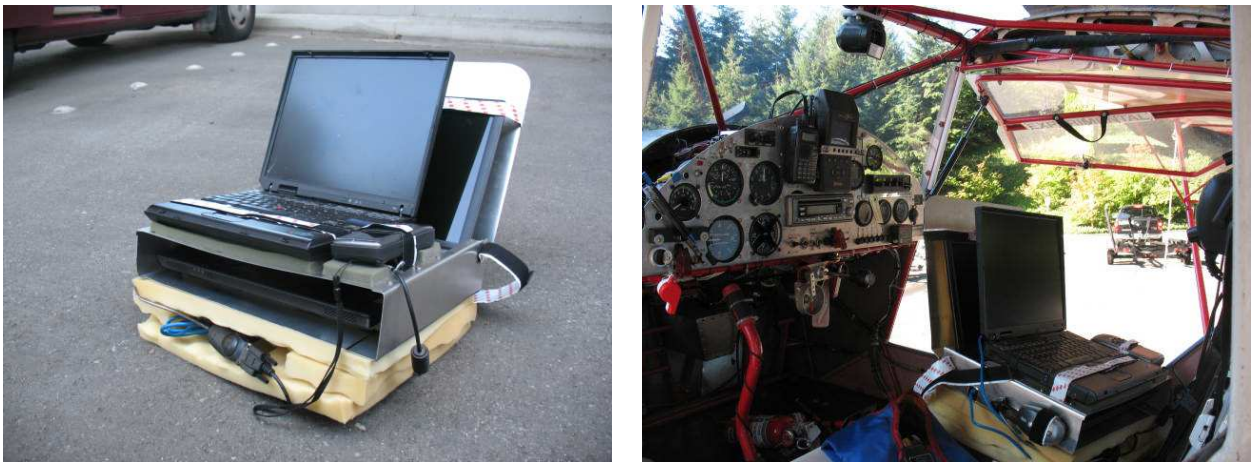


Figure 8. Flight test hardware connection diagram.



(a) Flight test hardware in protective carrier.

(b) Hardware situated in co-pilot seat.

Figure 9. Flight test hardware in test vehicle.

2. Flight Test Vehicle

The flight test was conducted using a 1993 Kitfox Classic IV. This is an experimental, two-seater, single engine aircraft with the performance specifications summarized in Table 2.

The aircraft can be equipped with either wheels or floats for amphibious operation. A picture of the aircraft is shown in Figure 10.

B. Mission Description

The flight test mission is modeled after a standard maritime search and rescue operation.[20] A single target is assumed to be lost somewhere at sea[21] and there are multiple agents searching the area for this target. In this scenario, the target test area is situated off the north east coast of Bainbridge Island, WA. A georeferenced image which covers the test area is obtained as shown in Figure 11(a). From this, the areas of water and land are assigned different colors as shown in Figure 11(b). The base of operations is designated as 47° 36' 1.46"N and -122° 32' 17.10"E. From the base of operations, a 6km by 6km test area is projected with part of the area over land and part over water. The test area is shown in Figure 11(b) as the red outline. Figure 11(b) is used to create an initial occupancy map[7,22] representation of the test area.

The mission involves searching the test area for the target using five agents. Four of the agents are

Table 2. Kitfox Classic IV specifications.

Specification	Value	Specification	Value
Manufacturer	Skystar	Fuel Capacity	26 gallons
Model	Kitfox Classic IV	Fuel Burn	5.2 gal/hr @ 5600 RPM
Year	1993	Cruise Speed	75 MPH
Tail Number	N328ML	Powerplant	Rotax 582-LC (65 Hp)
Serial Number	DCU033	Propeller	Ivoprop adjustable pitch
Top Speed	125 MPH	MAC	51.1”
Range	300 Miles	Max Rate of Climb	600 ft/min
MGTOW	1200 lbs	Service Ceiling	14,000 ft
Dry Weight	769 lbs	V_y (max dy/dt)	63 MPH @1089 lbs
Wingspan	32 feet	V_x (max dy/dx)	56 MPH @1089 lbs
Wing Area	130.5 ft^2	V_{stall}	51 MPH
Aspect Ratio	7.85		

Figure 10. 1993 Kitfox Classic IV flight test vehicle.

simulated and one is the real aircraft. Two of the simulated agents have parameters used to emulate the ScanEagle unmanned aerial vehicle[23] and two have parameters to emulate the SeaFox unmanned surface vehicle.

C. Flight Test Results

Once the aircraft enters the test area, the algorithm is started to guide the aircraft and the four simulated agents in the search mission. The mission time limit is set for 20 minutes. Each agent’s path is updated once every 60 seconds and the visualization output to the operator of the aircraft is updated once every 4

(a) Georeferenced image of Puget Sound

(b) Simplified image with test area in red

(c) Initial occupancy map of flight test area

Figure 11. Flight test area of interest.

seconds. In this flight test, the co-pilot is in charge of monitoring the output of the autonomous algorithm and then relaying the information to the pilot. During the flight test, the next waypoint was conveyed to the pilot by communicating a heading angle and distance to next waypoint (there was no wind during the flight test so heading angle was approximately equivalent to course angle). The trajectory for the entire mission for the real agent is show in Figure 12.

In Figure 12, the red crosses represent the actual position of the agent as recorded by the GPS. The purple circles show the agent's position projected onto the test domain. The agent's position is projected onto the search area. The only difference between the trajectories is when the agent leaves the test area. Theoretically, this will never happen as the algorithm will only generate paths which are within the search area. This error only occurs when the pilot does not follow the desired path accurately enough.

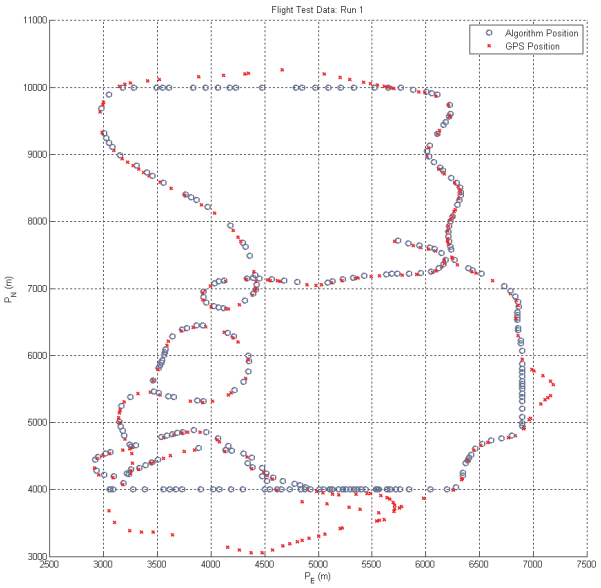
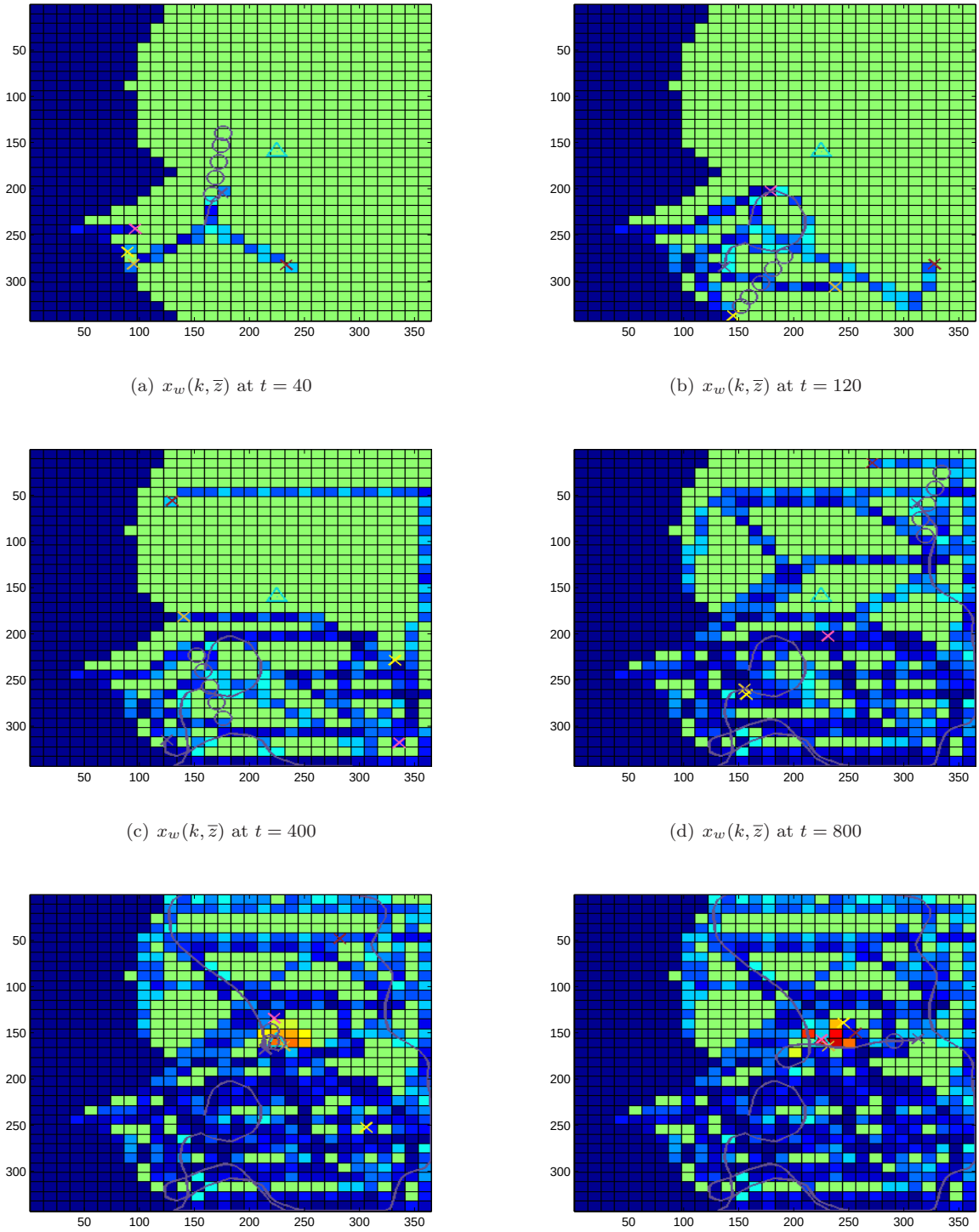


Figure 12. GPS and recorded agent position during flight test.

The same run and the associated occupancy based map and the other agents is shown in Figure 13.

In Figure 13, the purple cross represents the location of the real agent. The red, gold, yellow, and magenta crosses represent the location of the simulated agents. The purple circles are the paths planned for the real agent. The solid purple line is the real agent's trajectory. To avoid cluttering the figure, the associated paths and trajectories for the simulated agents are omitted. The target is assumed to be static (i.e. a boat that has lost power in standing waters) and its location is represented by the teal triangle. It is worth mentioning that the flight test in Figure 13 is the first time this particular pilot had used this system besides practice runs on the distributed simulator. This is evident in Figure 13(b) which shows some initialization errors on the part of the pilot as he becomes accustomed to the interface (initially, the pilot is unable to track the path). Figure 13(d) shows the pilot successfully tracking the path specified by the algorithm. Eventually, one of the agents locates the target and updates cells in the local area. This causes nearby agents, including the real agent, to converge on the target location (Figure 13(e)) and terminate a successful mission. Note that at this point, the strategic searching mission is terminated and control can be transferred to a tactical level autonomous algorithm which might be in control of tasks such as target tracking[24,25] and observation.

Note that the consequence of introducing a human operator to act as an inner loop controller is evident in the real agent's trajectory. Using an architecture where the pilot is responsible for dynamics and control



(a) $x_w(k, \overline{z})$ at $t = 40$

(b) $x_w(k, \overline{z})$ at $t = 120$

(c) $x_w(k, \overline{z})$ at $t = 400$

(d) $x_w(k, \overline{z})$ at $t = 800$

(e) $x_w(k, \overline{z})$ at $t = 1040$

(f) $x_w(k, \overline{z})$ at $t = 1160$

Figure 13. Flight test results with 4 simulated agents and 1 real agent.

level tasks (Figure 3) instead of the fully autonomous system (Figure 2) yields a decrease in performance. Many times during the actual flight test, the real agent was not on the path specified by the algorithm. This was due to several factors. The most significant of these was the fact that the time between visualization

updates was relatively large. Although the path for each agent was valid for 60 seconds, the agent's position and the pilot visualization screen were only updated at a rate of 0.25 Hz. Furthermore, there were only three updates between waypoints. If the agent was off course, the pilot would first be made aware of the error at the first update. The pilot would then only have two further feedback updates to correct the problem. The 4 second delay between updates proved problematic because the cross track error may grow to a significant value before the pilot is first made aware of the error. The dynamics of the aircraft were slow enough to exacerbate this problem. In other words, the maximum yaw rate was not sufficient to correct the cross track error quickly. The overall effect was that the pilot was constantly trying to correct errors and would not converge to the desired waypoint until near the end of the path. Despite this, the algorithm is robust in the sense that it gracefully handles these errors and still performs the search successfully.

VI. Conclusion and Further Research

This paper presented an architecture for verifying and validating the performance and output of strategic control algorithms with a high degree of accuracy while minimizing time between simulation and flight test. This was done by introducing human interaction at specific points in the system which preserves the autonomous contributions of the strategic algorithm by reducing the human to a simple inner loop controller. This architecture is used in the distributed ground based simulator to simulate flight test conditions in a controlled environment. The ground based distributed simulator is used to verify and validate the strategic control algorithms and also to familiarize pilots with the interface and to train them before actually performing a flight test.

During the development of the distributed simulator, many different methods of allowing human interaction were experimented with. Significant development effort was focused on developing a pilot visualization system which relays the pertinent path information to the pilot. By interviewing pilots after they had used the system, it was discovered that most of them only rely on the overview of the path (the bottom graph in Figure 5(b)). The operator workload became too high when they were forced to constantly scan all four displays and process the information. Current research is directed towards developing a single display which efficiently conveys all the relevant information to the pilot.

The other major interface between the simulator and the pilot involves the joystick. The current joystick is only a three axis device. The two principal axes were mapped to elevator and aileron inputs. Initially, the third axis was mapped to the throttle input and the rudder was fixed to zero. This design was selected because some commercial pilots expressed that rudder input is typically reserved for a yaw damping system. However, it was discovered that path following was easier for the pilot if sideslipping and coordinated turns were allowed. To facilitate these maneuvers, the third axis was mapped to the rudder input. The throttle was instead controlled using the buttons and an integrator scheme.

Finally, the current joystick automatically re-centers the two main axes to zero when the pilot takes their hands off the stick. Some pilots were displeased by this because it made it difficult to trim the aircraft during a run. The next generation simulator will include a more sophisticated joystick with more than three axes and trim features.

All of the functionality described previously is readily ported to 3 dimensional examples with minimal changes.

VII. Acknowledgements

This work is sponsored in part by the Washington Technology Center (grants F04-MC2 and F04-MC3), the Osberg Family Trust Fellowship, the Washington NASA Space Grant Consortium, and the Air Force Office of Scientific Research (grant FA-9550-07-1-0528). The authors would also like to thank other members of the Autonomous Flight Systems Laboratory, Dr. Anawat Pongpunwattana and Dr. Richard Wise for contributions to the simulator. The authors are also grateful to the flight test pilot, Matthew Lum, for his contributions and input to the project.

References

[1]Antisaklis, P. J. and Passino, K. M., "Towards Intelligent Autonomous Control Systems: Architecture and Fundamental Issues," *Journal of Intelligent and Robotic Systems*, Vol. 1, No. 4, 1989, pp. 315–342.

[2]Clough, B. T., "Metrics, Schmetrics! How the Heck Do You Determine a UAV's Autonomy Anyway?" *Proceedings of the Performance Metrics for Intelligent Systems Workshop*, Gaithersbug, MD, 2002.

[3]Huang, H.-M., Messina, E., Wade, R., English, R., Novak, B., and Albus, J., "Autonomy Measures for Robots," *Proceedings of the International Mechanical Engineering Congress*, 2004.

[4]Sholes, E., "Evolution of a UAV Autonomy Classification Taxonomy," *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, 2007.

[5]Johnson, E. N., Rooz, N., Hur, J., and Pickell, W., "A Concurrent Testing Process for Research Unmanned Aerrial Vehicles," *Proceedings of the 25th AIAA Aerodynamic Measurement Technology and Ground Testing Conference*, San Francisco, CA, June 2006.

[6]Proctor, A. A., Kanna, S. K., Raabe, C., B., C. H., and Johnson, E. N., "Development of an Autonomous Aerial Reconnaissance System at Georgia Tech," *Proceedings of the Association of Unmanned Vehicle Systems International Unmanned Systems Symposium and Exhibition*, 2002.

[7]Lum, C. W., Rysdyk, R. T., and Pongpunwattana, A., "Occupancy Based Map Searching Using Heterogeneous Teams of Autonomous Vehicles," *Proceedings of the 2006 Guidance, Navigation, and Control Conference*, Keystone, CO, August 2006.

[8]Lum, C. W., Rysdyk, R. T., and Pongpunwattana, A., "Autonomous Airborne Geomagnetic Surveying and Target Identification," *Proceedings of the 2005 Infotech@Aerospace Conference*, AIAA, Arlington, VA, September 2005.

[9]Pongpunwattana, A., Wise, R., Rysdyk, R. T., and Kang, A. J., "Multi-Vehicle Cooperative Control Flight Test," *Proceedings of the 25th Digital Avionics Systems Conference*, October 2006.

[10]Lum, C. W. and Rysdyk, R. T., "Time Constrained Randomized Path Planning Using Spatial Networks," *Proceedings of the 2008 American Control Conference*, Seattle, WA, June 2008.

[11]Anderson, D. E. and Pita, A. C., "Geophysical Surveying with GeoRanger UAV," *Proceedings of the 2005 Infotech@Aerospace Conference*, The Insitu Group, Arlington, VA, September 2005.

[12]Lum, C. W., Rowland, M. L., and Rysdyk, R. T., "Human-in-the-Loop Distributed Simulation and Validation of Strategic Autonomous Algorithms," *Proceedings of the 2008 Aerodynamic Measurement Technology and Ground Testing Conference*, Seattle, WA, June 2008.

[13]Lambrechts, P., Bennani, S., Looye, G., and Helmersson, A., "Robust Flight Control Design Challenge Problem Formulation and Manual: the Reserach Civil Aircraft Model (RCAM)," Tech. rep., Group for Aeronautical Research and Technology in Europe, Europe, 1997.

[14]Lum, C. W. and Vagners, J., "A Modular Algorithm for Exhaustive Map Searching Using Occupancy Based Maps," *To appear in Proceedings of the 2009 Infotech@Aerospace Conference*, Seattle, WA, April 2009.

[15]"FlightGear Flight Simulator," Public Information, 2006, http://www.flightgear.org/.

[16]Rysdyk, R. T., "Unmanned Aerial Vehicle Path Following for Target Observation in Wind," *Journal of Guidance, Control, and Dynamics*, September 2006, pp. 1092–1100.

[17]Unmanned Dynamics, Hood River, OR, *AeroSim Aeronautical Simulation Blockset User's Guide Version 1.2*.

[18]Vincenty, T., "Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations," *Survey Review*, Vol. 176, 1975, pp. 88–93.

[19]"Department of Defense World Geodetic System 1984, Its Definition and Relationships with Local Geodetic Systems 3rd Edition," Tech. Rep. TR8350.2, National Geospatial-Intelligence Agency.

[20]Lum, C. W., *Coordinated Searching and Target Identification Using Teams of Autonomous Agents*, Ph.D. thesis, University of Washington, Seattle, WA, March 2009.

[21]Bourgault, F., Furukawa, T., and Durrant-Whyte, H., "Coordinated Decentralized Search for a Lost Target in a Bayesian World," *Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Australian Centre for Field Robotics, Las Vegas, NV, October 2003.

[22]Elfes, A., "Using Occupancy Grids for Mobile Robot Perception and Navigation," *IEEE Computer*, 1989, pp. 46–57.

[23]"ScanEagle UAS Flies with Heavy Fuel in Iraq," Insitu Press Release, http://www.insitu.com/index.cfm?navid=20&cid=2582.

[24]Klein, D. J., *Coordinated Control and Estimation for Multi-agent Systems: Theory and Practice*, Ph.D. thesis, University of Washington, Seattle, WA, September 2008.

[25]Rysdyk, R. T., Lum, C. W., and Vagners, J., "Autonomous Orbit Coordination for Two Unmanned Aerial Vehicles," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, San Francisco, CA, August 2005.