7-2008

Wide-Area Surveillance System using a UAV Helicopter Interceptor and Sensor Placement Planning Techniques

Marcus James Jackson
University of Tennessee, Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Marcus James Jackson entitled "Wide-Area Surveillance System using a UAV Helicopter Interceptor and Sensor Placement Planning Techniques." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Mongi Abidi, Major Professor

We have read this thesis and recommend its acceptance:

Andreas Koschan, Seddik Djouadi

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Marcus James Jackson entitled “Wide-Area Surveillance System using a UAV Helicopter Interceptor and Sensor Placement Planning Techniques.” I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Mongi Abidi,

Major Professor

We have read this thesis
and recommend its acceptance:

Andreas Koschan

Seddik Djouadi

Accepted for the Council:

Carolyn R. Hodges,

Vice Provost and
Dean of the Graduate School

(Original signatures are on file with official student records.)

Wide-Area Surveillance System using a UAV Helicopter Interceptor and Sensor Placement Planning Techniques

A Thesis

Presented For The

Master of Science

Degree

The University Of Tennessee, Knoxville

Marcus James Jackson

July 2008

Abstract

This project proposes and describes the implementation of a wide-area surveillance system comprised of a sensor/interceptor placement planning and an interceptor unmanned aerial vehicle (UAV) helicopter. Given the 2-D layout of an area, the planning system optimally places perimeter cameras based on maximum coverage and minimal cost. Part of this planning system includes the MATLAB implementation of Erdem and Sclaroff's Radial Sweep algorithm for visibility polygon generation. Additionally, 2-D camera modeling is proposed for both fixed and PTZ cases. Finally, the interceptor is also placed to minimize shortest-path flight time to any point on the perimeter during a detection event.

Secondly, a basic flight control system for the UAV helicopter is designed and implemented. The flight control system's primary goal is to hover the helicopter in place when a human operator holds an automatic-flight switch. This system represents the first step in a complete waypoint-navigation flight control system. The flight control system is based on an inertial measurement unit (IMU) and a proportional-integral-derivative (PID) controller. This system is implemented using a general-purpose personal computer (GPPC) running Windows XP and other commercial off-the-shelf (COTS) hardware. This setup differs from other helicopter control systems which typically use custom embedded solutions or micro-controllers.

Experiments demonstrate the sensor placement planning achieving >90% coverage at optimized-cost for several typical areas given multiple camera types and parameters. Furthermore, the helicopter flight control system experiments achieve hovering success over short flight periods. However, the final conclusion is that the COTS IMU is insufficient for high-speed, high-frequency applications such as a helicopter control system.

Contents

List of Tables

List of Figures

1. Introduction

Many sensitive areas, such as armories, hazardous materials storage, military bases, borders, etc., require complete wide-area surveillance and monitoring. However, significant manpower is needed to adequately monitor these large outdoor environments. A viable solution should provide persistent wide-area surveillance at an optimized cost. Furthermore, the system should have the capability to detect and track intruders that enter the restricted area. One viable solution utilizes automated aerial surveillance through unmanned aerial vehicles (UAV).

In this system, perimeter video cameras (or human guards) are responsible for persistent monitoring (motion, intruder detection, etc.). In the event an intruder is detected, an alarm is activated while the intruder is automatically tracked via live video. To ensure the highest reliability of intruder tracking, an autonomous UAV with intelligent imaging capabilities can "intercept" and track the intruder until apprehended. Therefore, this thesis describes the development of such a system based on a UAV helicopter platform, also expressed further on as the "robot," "vehicle," "UAV," or simply "helicopter."

The first part of this work focuses on methods of sensor and interceptor placement planning towards a cost-optimized persistent perimeter monitoring system. The second part describes the design and implementation of the UAV helicopter-interceptor system.

1.1 Motivation

As one of the most successful methods of surveillance in oversea conflicts, unmanned vehicles, vehicles that are operated remotely, have recently received significant attention, particularly unmanned aerial vehicles (UAV). The primary reason for this growth of interest is due to the unmanned vehicle's ability to perform the dangerous, dull, or dirty (DDD) missions instead of a human operator.

A "dangerous" mission is obvious – one that could potentially endanger the life of the pilot and crew. A "dull" mission is usually long, tedious, and routine. Essentially, "dull" missions require constant attention and repetitive interaction from the pilot. Finally, a "dirty" mission typically deals with disasters or hazards. For example, in the past, when the United States was testing nuclear weapons by detonating them offshore, a single fighter jet with a human pilot would fly into the fallout area after detonation to collect data. An unmanned vehicle could perform missions such as these much more safely and efficiently.

Fixed-wing vehicles (such as the Predator) require a runway to take-off and land, and cannot stop in mid-air. They must instead loiter, flying circles or figure-eights around a stationary target. Rotary wing vehicles (helicopters), on the other hand, have the distinct advantages of vertical take-off and landing (VTOL) and hovering in a fixed position. The payload capacity is also typically larger than that of a comparable fixed-wing vehicle.

Helicopters provide an especially interesting and useful platform due to their ability to perform two different modes of flight: hovering and forward-flight. While hovering, the controller of the helicopter must balance it against gravity and in six-

degrees of freedom: up, down, north, south, east, and west. The end result is a vehicle that is able to become stationary at any altitude or position.

This behavior is especially useful in surveillance, when an interesting object might need to be observed with a fine amount of detail. In addition, a helicopter is able to hover slowly in any direction. While fixed-wing aircraft must maintain a minimum airspeed to stay in the air, helicopters are able to drift slowly about, providing a potentially higher level of detail and resolution of in-flight data.

One application where hovering becomes vital is object tracking. When a person or object enters a restricted area and must be followed, the helicopter must be able to deal with drastic changes of speed, from stopped completely to its maximum speed. Thus, the helicopter provides the means to track an unpredictable object with dynamic velocity.

While in forward-flight, the helicopter tilts off-center and accelerates from a stable hover. As the velocity of the helicopter increases in a lateral direction, the rotors of the aircraft begin to behave less like rotating rotors and more like a fixed-wing. Thus, the helicopter actually behaves more like an airplane during flight. The ability of forward-flight allows for increased airspeed and different flight dynamics. Additionally, tracking and following of high-speed objects becomes applicable with forward-flight. The helicopter is also provided the ability to move point to point quickly during time-critical missions.

These two modes of flight are not completely distinct. Rarely is the helicopter in full forward-flight; rather, the helicopter's flight behavior gradually moves from that of a rotor vehicle in hover to a fixed-wing vehicle flying forward. External conditions affect the behavior as well, particularly wind and temperature. Helicopter flight is a dynamic and nonlinear operation.

In terms of payload, an aerial vehicle can be equipped with an array of sensor packages. Navigational sensors include gyroscopes, magnetometers, accelerometers, altimeters, and global positioning systems (GPS). Data collection sensors generally include video cameras, still cameras, thermal imaging systems, laser-range scanners, and other high-tech imaging devices.

1.2 Problem Description

Large outdoor areas, particularly sensitive borders and military bases, require the highest level of persistent monitoring possible. Monitoring includes the detection of intruders at all points on the perimeter. Thus, the problem is constant surveillance of large, closed environments with the ability to track detected intruders. Intruders can enter at any point on the perimeter and must be intercepted in minimal time. This problem is general, and could have many solutions. Here, we propose a cost-minimizing method employing simple perimeter camera emplacements and a UAV helicopter interceptor.

A complete diagram of an envisioned wide-area camera surveillance system with a UAV helicopter interceptor is depicted in Figure 1. Here, the dotted-line ovals represent omni-directional camera coverage along the fence-line. The helicopter interceptor, located at the light blue square, is responsible for intercepting intruders detected by the cameras at the perimeter. Actual experiments are extended to include entire interior coverage along with both fixed and PTZ camera modeling.

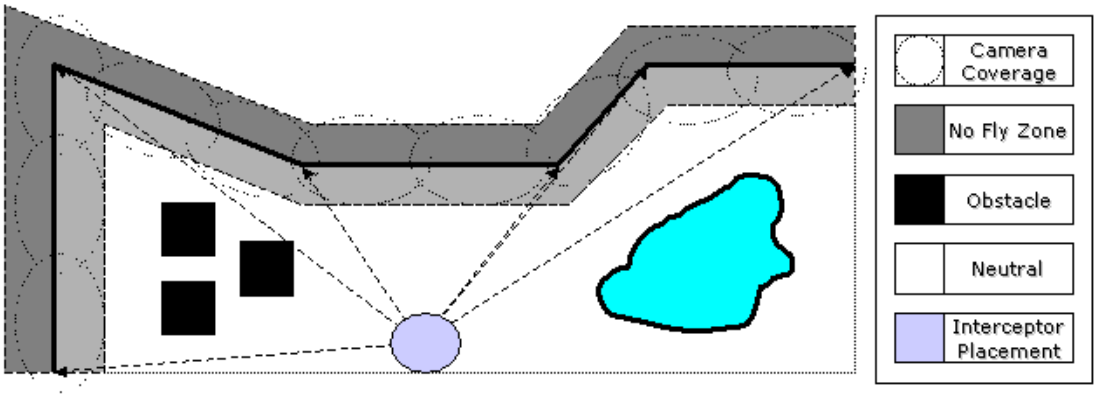


Figure 1. Complete UAV helicopter interceptor system.

"Interception" essentially entails the deployment of an interceptor robot to a location within line-of-sight of the intruder. The helicopter is not expected to physically intercept the intruder, but rather to track him or her. Thus, the overall system is a two-step detection-interception model, where perimeter camera emplacements would handle the persistent monitoring and initial detection. A positive detection would instruct the interceptor to navigate to an initial location and track the intruder from there.

The helicopter control system itself is therefore the focus of this thesis. Because the goal is completely automated surveillance, the helicopter must possess autonomous navigation abilities for intruder interception. Autonomous helicopter navigation begins with simple hovering in place. Hovering can be extended to allow for controlled point-to-point translation. However, for fastest intruder pursuit, the helicopter's forward-flight dynamics require a more complex control system.

1.3 Contributions

The primary contributions of this thesis include:

1. The design of a flight control system for the Imaging, Robotics, and Intelligent Systems (IRIS) miniature UAV helicopter, which is normally manual-controlled by a human R/C pilot.
2. Implementation of the flight control system using commercial off-the-shelf (COTS) parts and a general-purpose PC (GPPC) running Windows XP (as opposed to custom embedded systems or microcontroller-based solutions).
3. Design and implementation of a sensor and interceptor placement planning system towards wide-area surveillance.

Many similar projects have built the entire system from the ground-up. Here, the goal is to use existing commercial products – hobby-grade R/C helicopter, general-purpose PC running Windows XP, etc. – to create a semi-autonomous UAV helicopter

system. Thus, many operations are simply written into the software, and most communication is through standard interfaces (i.e., USB).

1.4 Organization

Chapter 2 comprises the background literature survey performed while researching multi-robot control systems towards automated surveillance. Chapter 3 describes the sensor placement planning and control system design theory. Chapter 4 outlines the individual experiments performed during the incremental design. Finally, Chapter 5 concludes the paper.

2. Literature Review

The literature review first looks at other current multi-robot systems towards surveillance-type applications. Next, UAV flight control systems are examined. Finally, the chapter concludes with a brief look into wireless R/C communications.

2.1 Multiple Unmanned Vehicle Control Systems

A multiple unmanned vehicle (UV) or robot control system can be broken down into five parts: high-level control, low-level control, communications, sensing, and central control. The high-level control encompasses the mission planning and allocation systems. The low-level control is the basic navigational system of the robot. The communications is the method that the robot coordinates and interacts with other robots in the system. The sensing system includes the methods that the robot samples its environment and builds a model of the world. Finally, the central control system provides a monitoring and control interface as well as a central database accessible by all robots. Multiple-robot control systems seek to automate the process of coordinating a large group of robots by using a high-level command strategy and allowing autonomous robots in the field to plan, coordinate, and act on their own.

Previously, the common trend in these types of systems was toward the high-level control, mission planning and allocation, being performed by the central control station. Recently, however, more research has focused on distributed intelligence, allowing individual robots to operate fully autonomously, coordinating with other robots, planning, and acting on their own. The autonomous robot is still able to accept missions given to it by the central control station, but is able to dynamically formulate and re-plan missions along the way.

In the majority of these systems, the individual low-level control systems of the robots are abstracted. Most simply consider the onboard controller to accept waypoint commands, as is common in many autonomous navigation controllers. Some systems also consider the robot to be able to perform simple tasks, such as object avoidance. However, other problems such as trajectory-tracking, preventing two air vehicles from colliding, are usually considered high-level and handled appropriately.

The communications structure in a multi-robot system is of utmost importance. When a robot of low-level intelligence is cut-off from the rest of the system, it might not be able to complete its mission without further input, or even have a mechanism for returning home. The communications in a complex, multi-node, low-bandwidth network such as these is a popular research area.

Some of the most notable systems of late are described in the following sections.

2.1.1 Emerging Results in Cooperative UAV Control

Ryan, *et al.* [2004] with the Office of Naval Research (AINS) performed a brief survey on emerging results in cooperative UAV control. The focus was on current research in cooperative UAV control, such as efficient computer vision for real-time navigation, networked computing, communication strategies for distributed control,

collision avoidance, and formation flight. Without reiterating the details of that survey, the five main topics were:

Aerial Surveillance and Tracking: examining the issues in small, fixed-wing surveillance packages, including size/weight limitations and real-time processing capabilities.

Collision and Obstacle Avoidance: one of the largest dangers multiple-UAV systems is the chance of two friendly vehicles colliding accidentally. Research into this topic considers the limitations of onboard vision processing along with the popularity of GPS navigation. Solutions include trajectory tracking and better communication/cooperation among UAVs. For obstacles, UAVs must be trained to "see" the terrain or have an onboard terrain map.

Formation Reconfiguration: formation flight is a solution to the collision problem, as well. However, reconfiguring the formation during a flight is a problem in itself. The paper presents both fixed- and flocking-formation solutions for dynamic collision/obstacle avoidance.

High-Level Control: this section is most applicable to the survey presented here. High-level control includes a user interface, communications framework, and the logic to convert mission-level commands into resource allocations and formation assignments. Additionally, the fusion of data from multiple vehicles must be combined to form a map of the world. Thus, the system must be modular in order to decompose the large control problem and included a wide variety of implementation platforms (sensors and UAVs).

Hardware and Communication: finally, the last section of the survey considers the hardware topics with the latest small, low-cost aircraft used in multi-UAV programs, today. Payload weight and size, as well as the endurance of the vehicles, are issues here. Additionally, issues with vision systems, long-range, out-of-LOS communication, and modern flight control systems are examined.

2.1.2 COMETS

The COMETS (Real-Time Coordination and Control of Multiple Heterogeneous Unmanned Aerial Vehicles) project described in Ollero, *et al.* [2005] seeks to utilize airships and helicopter vehicles in a cooperative multi-vehicle environment. The main application of COMETS is the detection, monitoring, and fighting of forest-fires [Merino, 2005].

The system assumes multiple levels of robot intelligence – some units are directly controlled by a human operator, some might have operational autonomy (the ability to navigate without the ability to plan), while others perform fully autonomously. Thus, the architecture of the system requires the integration of both central and distributed decision-making schemes.

With the infinite variety of robot intelligence, the system should be able to dynamically configure itself according to the decisional capabilities of each individual robot. Thus, a generic supervisor, a plugin controller, is assigned dynamically to each robot. The generic, plug-in supervisor modules are stored in database tables and based on models of the vehicles. Thus, the models can be tuned over time to the dynamics of each vehicle and also its resource consumption.

The individual decisions within this multi-robot system consider a few different ideas:

Supervision and execution: the executive is the reactive management during the execution of a task. The supervision of a task is active control of the decisional activities of a robot at all times.
Coordination: ensures the multi-robot cooperative task execution. Solves conflicts between robots, whether they are physical (trajectory-based) or resource-conflicts.
Mission refinement, planning, and scheduling: dedicated to the creation, planning, refinement, and scheduling of tasks. In addition, creates the models of the world based on current knowledge, such as the motion and perception of the robot.
Task allocation: distributes tasks among each robot. Considers to capabilities of each robot and the relevance to other current tasks.

The system is broken up into three segments: ground (the central control center), flying (each individual helicopter or airship control system), and communications. The ground segment performs the centralized planning of the system through the Mission Planning System (MPS), as well as proving an interface for human operators through the Monitoring and Control System (MCS). Also, cooperative environment perception tasks are performed within the ground segment through the Perception System (PS).

Within the ground segment is the control center – the core of the decisional system. The control center provides a graphical user interface (GUI) to human operators, as well as high-level control (deliberative) of vehicles within the system. The function of the control center is to break down abstract mission plans into sequences of atomic procedures executable by the vehicles, and provide real-time monitoring and control of the mission execution.

Figure 2 depicts the global architecture of the COMETS system [Ollero, 2005]. The break down of a mission within the COMETS system includes the basic topics, such as mission decomposition, resource allocation, path planning, and conflict resolution. The end goal is to provide a timeline of low-level procedures for each vehicle without inducing any conflicts in the system.

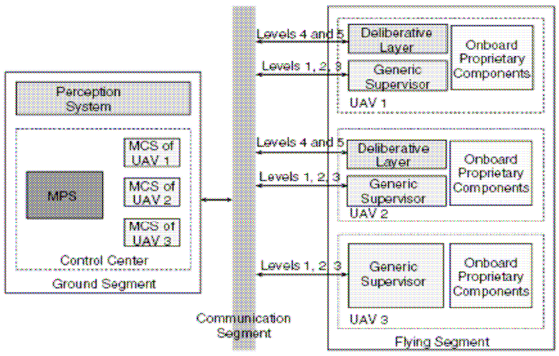


Figure 2. COMETS Global Architecture [Ollero, 2005].

The flying segment encompasses all unmanned air vehicles in the system. Each robot has onboard proprietary components, such as flight control, data acquisition, and data processing. A generic supervisor is defined that interfaces the vehicle to the COMETS system, as well as controlling the vehicle. Above the supervisor sits a deliberative layer, if the robot has fully autonomous capability, which performs the high-level control of the robot – the mission planning, refining, and scheduling.

The intelligence of each individual robot is broke into five levels within the COMETS system [Gancet, 2004]:

Level 1: there is no autonomy onboard the robot. Elementary from only the control center tasks can be received and executed.
Level 2: the robot has executive capabilities, also known as operational autonomy. At this state, the robot can accept sequences of fundamental tasks and act upon them. The status of the operations is returned to the control center.
Level 3: at this stage, the robot has all the capabilities of level 2, plus interacting capabilities with other robots of the same level of intelligence. For example, the robots can dynamically synchronize between one another.
Level 4: Beginning with level 4 is the introduction of high-level intelligence. Task requests are managed onboard and can be planned and scheduled. At this stage, the deliberative intelligence of the overall system becomes distributed among the high-level intelligence robots, instead of only at the control center.
Level 5: Finally, level 5 constitutes full autonomy. The robot has all of the task management abilities of level 4 plus the ability to reallocate tasks among its peers.

Up to intelligence level 3 (low-level), the control center handles the global consistency of the system through mission planning. By levels 4 and 5, the coordination and mission planning can be distributed among the high-level intelligence vehicles. A table categorizing the levels of intelligence presented above is presented on the following page.

High-level intelligence vehicles can include onboard task-planning and scheduling, coordination, and reallocation subsystems. The deliberative layer builds executable plans for each vehicle. However, during execution, it might be required to dynamically change the plans to better fit the mission. Thus, the original plan should be able to be processed again online, according to the current situation. Coordination, as mentioned above, is both spatial and interactions-related. Spatial coordination is crucial – two vehicles physically striking each other could be disastrous. Work is still ongoing in negotiation and coordination of non-conflicting planning. Finally, task reallocation is a possibility for level 5 intelligence vehicles. This behavior would allow similar tasks to be shared to a single vehicle, and more capable vehicles could dynamically allocate suitable tasks when needed for more efficient operation. Table 1 depicts the various levels of intelligence [Gancet, 2004].

Table 1. Levels of intelligence and partitioning between a central station and a robot [Gancet, 2004]

		Supervision and execution	Coordination	Planning	Task allocation
Level 1	Central	X	X	X	X
	Distrib.	-	-	-	-
Level 2	Central	x (supervision)	X	X	X
	Distrib.	x (executive)	-	-	-
Level 3	Central	x (supervision)	x (high level coordination)	X	X
	Distrib.	x (executive)	x (low level coordination)	-	-
Level 4	Central	-	-	-	X
	Distrib.	X	X	X	-
Level 5	Central	-	-	-	-
	Distrib.	X	X	X	X

Tasks in the COMETS system are built using elementary, atomic subtasks [Ollero, 2004]. The tasks include take-off (TO), go-to (GT), take-shot (TS), wait (WT), and land (LD). The atomic tasks are self-explanatory, with the exception of "take-shot" that means to perform a perception action (i.e., take a picture). Tasks can be inserted into the system dynamically in four different modes: sequential (SEQ), very urgent task (VUT), dependant (DEP), and non-urgent task (NUT). SEQ mode tasks are inserted in a sequence of tasks with pre- and post-conditions, which can be mandatory or optional. Mandatory tasks must be satisfied for the SEQ mode task to execute. Optional tasks can be considered satisfied if they complete, or even if they find themselves "un-satisfiable." After execution, the post-condition must be satisfied, as well. VUT mode is a mechanism for executing a priority task immediately. DEP mode is similar to a sequential task, but can have multiple mandatory preconditions that must be satisfied before it can execute. Finally, NUT mode is similar to DEP mode in the fact that it can have multiple preconditions, but they can be optional. The full task planning and control system is described in detail in [Gancet, 2005].

The communications system (CS) creates network nodes (NN) out of tasks in either single- or multi-tasking environments. For flexibility, a server-less, peer-to-peer approach was taken, and the network is able to sit on top of almost any transportation layer. Originally, the communications was based around the notion of a "Contract Net [Lamaire, 2004]." However, distributed shared memory, i.e. a blackboard (BB), was later chosen for data sharing. Each network node contains an internal copy of the BB. The BB is comprised of state-type and streaming "slots," where each slot has an assigned bandwidth, data type, and a specific amount of data. State-type slots contain the latest information (state) of the system, while streaming slots are generally for streaming sensor data (video or images). Mutual exclusion is enforced by the communications subsystem on all of the slots.

The perception system encompasses the application-independent image processing (AIIP), detection/alarm confirmation, localization, and evaluation service (DACLE), the event monitoring system (EMS), and the terrain mapping system (TMS). The AIIP contains a suite of necessary image processing packages, such as camera stabilization, position estimation, image geo-location, and object tracking. The details of the AIIP package are described in detail in [Ollero, 2004]. The technique is based on large-feature matching, as well as projective methods, and can be applied especially to helicopter motion compensation and object detection.

The DACLE subsystem is specific to the COMETS application of forest-fire fighting. The system serves to detect fires through IR and video images, and sound a detection alarm. The EMS subsystem deals with the monitoring activities of the COMETS system. When an event is detected, monitoring missions are scheduled and executed. The TMS subsystem uses the initially available cartographic files and updates them according to collected sensor data and the results of the AIIP subsystem.

One of the main goals of COMETS is to provide a cooperative perception strategy to increase the precision of the location of an event. Techniques to fuse both commensurate data (such as from two parallel stereo-vision cameras) and non-commensurate data (such as video and IR data) are defined within COMETS. Data registration, non-coincident sampling, and data resolution are keys to these techniques.

2.1.3 MICA

UC Berkeley's Mixed-Initiative Control for Automata Teams (MICA) architecture is an integrated solution to the problem of UAV navigation in potentially hostile environments. Large scale UAV control is divided into hierarchical, modular tasks. Thus, the architecture is adaptable to a wide variety of missions, strategies, and sensing platforms.

In [Rathinam, 2004], an architecture is presented to control a team of unmanned aerial vehicles (UAV) in coordinated searches for ground threats in a region, such as surface-to-air missiles (SAM), ground troops, artillery, etc. The system is of a modular design, where sensors and the strategies are coupled in the architecture. In addition, the "safe flight" [Rathinam, 2004] path planning algorithm is improved upon to allow sufficient time for imaging operations, as well as cope with the mobility limitations of the UAV. The safe flight algorithm decreases the forward progress of a UAV to allow for sufficient time to process sensor data. Path planning within this system is based around the use of "risk maps" – essentially, probability maps of an enemy's location compiled using Bayes' rule. This map is shared with all robots in the system.

The general system architecture is composed of three main components: the Team Manager, the UAV Managers, and the Sensor Information Processing Unit (contains the risk map, target distribution model, and the probability update module). The UAV manager acts as a generic supervisor layer to the Team Manager – offering high-level control of the UAV instead of elementary commands. The Sensor Information Processing Unit fuses and shares the information gathered by the sensors onboard each UAV in the system. Finally, the Team Manager coordinates and monitors the team members at a high-level. All vehicles in the system use the safe flight strategy. Figure 3 depicts the system diagram.

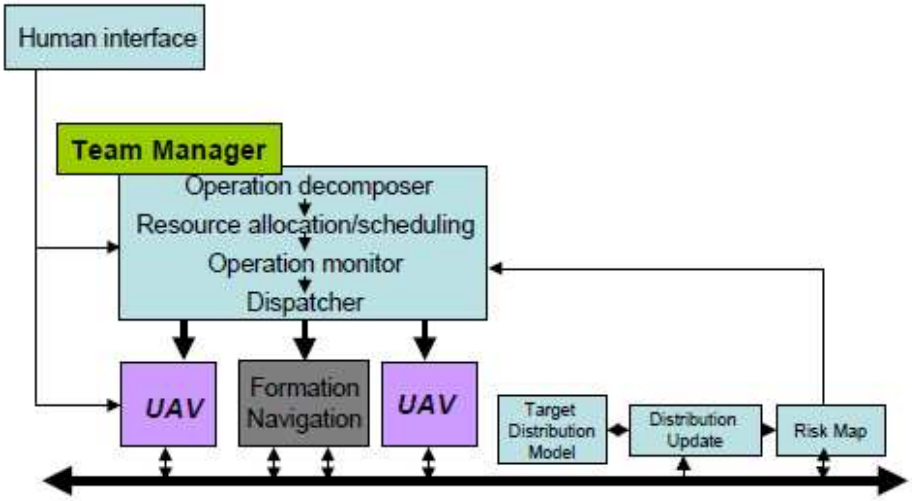


Figure 3. MICA System Architecture [Rathinam, 2004].

In the Team Manager, tasks can be given at a high-level, such as "search this area for threats." A mission can then be broke down into a series of tasks by the Operation Decomposer. Afterwards, the Resource Allocation unit allocates the necessary resources demanded by the mission. Next, the Dispatcher assigns the tasks to the allocated resources. Finally, the Operation Monitor is in place to monitor system status over the course of the mission. When a resource is lost (i.e., a UAV is destroyed), or one completes a mission and becomes available again, the Operation Monitor prompts the Resource Allocation unit to redistribute the task load using the latest set of resources. Thus, the system is dynamically adaptable to unforeseen conditions.

Flight control of a UAV is based on a Dynamic Path Planner (DPP) that sits on top of existing navigational control systems (i.e., autopilots). The DPP handles high-level control and also automatic obstacle avoidance. The DPP performs this task by creating minimum risk paths (using the risk map) between the current position and the destination waypoint. The path is created to avoid all obstacles as well as threat zones. During execution, the DPP interrupts if an obstacle or threat is detected and dynamically plans the path, again. Incidentally, for a fixed-wing UAV, the vehicle cannot stop in any one place to reformulate a plan – the loss in airspeed causes loss in altitude. Thus, the safe controller is in place to generate looping paths around a fixed point until the new nominal path is generated.

Currently, the system is implemented to perform strategic searching maneuvers (safe navigation between waypoints) and threat searching maneuvers (generating the threat map of an area). Again, both of these use the "safe flight" algorithm. The strategic searching uses the algorithm described above (in the section describing the DPP). For threat mapping, the UAV manager generates a space-filling curve, which is just a series

of waypoints, to map the designated area. Once complete, the UAV navigates using the minimum risk path from waypoint to waypoint, updating the risk map as it goes.
The MICA Open Experimental Platform was used to simulate the architecture presented above.

In [Sousa, 2004], an attack of Blue Force UAVs versus Red Force SAM sites and radars is designed. The attack is based on priori information and composed of a planning and execution phase. The planning phase selects targets, groups them into sub-tasks, allocates UAVs to these sub-tasks, and creates a risk-minimized path for the UAV to follow. The execution phase coordinates the UAVs using real-time controllers to achieve the given missions.

The implementation is created in the *Shift* programming language. *Shift* is essentially a language for describing networks of automata, and is briefly described in section III of [Sousa, 2004]. The execution control framework is composed of

- Controllers: individual controllers for each task, sub-task, sub-team, and UAV.
- Specifications: the modular tasks that the controllers execute, separated from control code.
- Localization: the location of a mobile controller is part of its state.
- Control Structure: the four-layer tree of controller nodes (task, sub-task, sub-team, UAV).
- Creation and Initialization: the control structure is built in steps from the root (task), with each controller creating its dependants and links to them.
- Adaptation: mobile controllers can adapt during a mission by having locations changed, being re-created to regenerate the control structure, being added or deleted upon initiation or completion of specifications, and having control dependencies changed.
- Patterns of coordination: each controller maintains coordination variables for each of its dependants. When it receives status updates from a dependant, it updates these variables and commands the dependants accordingly – allowing for distributed decision-making.

The Blue (air) force versus Red (ground) force scenario simulated here was based on one from the Boeing Open Experimental Platform using a *Shift* specification for the attack task.

Additionally, Berkeley's research includes hierarchial multi-agent, multi-modal systems [Koo, 2001]. In this system, bisimulation is the basis for the design of the hierarchy. Thus, the higher-level and low-level systems are similar. System specification conforms between its various levels of granularity by adopting a layered system to promote proof obligations. The approach is used to design a system controlling a group of autonomous agents in a pursuit-game versus multiple evaders. The target platform is UAV aircraft.

2.1.4 JAUS

The Joint Architecture of Unmanned (Ground) Systems (JAUS) [JAUS, 2005] is sponsored by the Office of the Under Secretary of Defense (OUSD) and was developed when the military realized the vast number of new robots being developed did not share a unified control system. In the field, a soldier is limited to carrying only a minimum number of equipment. Moreover, field troubleshooting and servicing the multiple varieties of robots, particularly when all require proprietary control mechanisms, could become impossible for a single technician. Thus, the JUAS standard was created to standardize the way robots communicate and are controlled.

JUAS is a component-based, message-passing architecture that specifies data formats and the methods of communication between multiple robots. The JAUS interface defines messages and behaviors that are independent of proprietary robot hardware. The primary goals of the JUAS architecture are to

1. Reduce life-cycle costs
2. Lower software maintenance costs
3. Lower training costs
4. Reduce the development time
5. Rapid prototype development
6. Rapid system engineering focused on new requirements
7. Create a framework for painless new technology insertion
8. Expand existing systems with new or better capabilities.

The domain model [JAUS Reference Architecture, 2005], depicted below, shows the various components of the JAUS architecture. Functional agents include the command, telecommunication, mobility, payloads, maintenance and training. Knowledge stores include the vehicle status, the world map, the library, and the log. Figure 4 depicts the JAUS domain model [JAUS Tutorial Presentation, 2005].

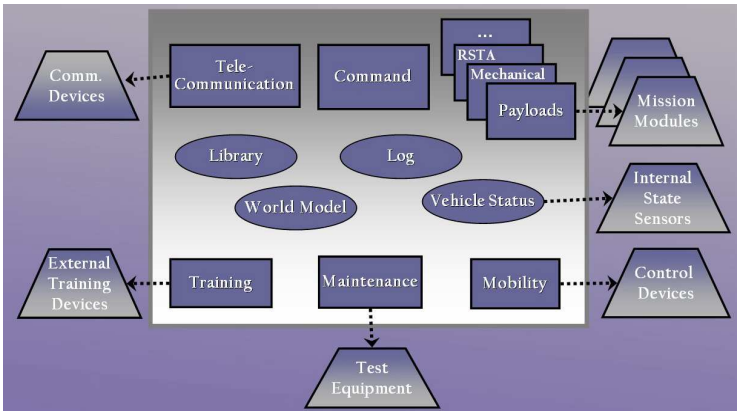


Figure 4. JAUS Domain Model [JAUS Reference Architecture, 2005].

The command agent collects data, makes decisions, and assigns tasks. A command agent can control a single robot, a squad (a team of robots), or a platoon (a team of teams of robots). The telecommunication segment manages connections between vehicles and the operator control unit (OCU). The payload agent manages the onboard payload – which could be munitions, an imaging package, etc. Maintenance maintains the system health, as well as interfaces with diagnostic equipment. Finally, training is responsible for providing training support for the operator either internally or using an external device.

The vehicle status knowledge store provides the real-time status of the vehicle to the rest of the system. The world map maintains the dynamically generated map of threats, terrain, obstacles, etc. The library keeps the reference material, procedures, and performance data on hand for evaluation. Finally, the log knowledge store keeps a running log of maintenance and training data.

While the domain model is considered more for the user of the JAUS system, the reference architecture [JAUS Reference Architecture, 2005] is meant for the scientist or engineer. The basic physical topology used within JAUS is in terms of Systems, Subsystems, Processing Nodes, and Components. A System is typically an OCU or entire vehicle. The next level, Subsystem, contains one or more processing nodes. A processing node has at least one CPU and one Message Routing Service (MRS). Finally, the processing node also contains Components, the individual atomic subsystems encompassed by a System. Message-passing can be performed within JAUS either periodically or aperiodically, allowing dynamic adjustment to changing bandwidth conditions.

JAUS is an excellent architecture for unmanned systems, exhaustively specifying every detail of message-passing between unrelated components. Only the briefest overview has been presented here, but all of the JAUS documentation can be found at http://www.jauswg.org/.

2.1.5 Stanford

Stanford has long been a leader in UAV research. Recent work has focused on a fully-capable multi-robot system consisting of fixed-wing UAV's (DragonFly [Teo, 2004]), as well as a testbed of rotorcraft vehicles (STARMAC [Hoffmann, 2004]). Both systems are autonomous and entirely developed by Stanford University, including the onboard architecture, avionics, flight control, and integration with wireless communications.

The fixed-wing aircraft used for the DragonFly project have approximately 10-foot wingspans and are of a traditional single propeller design. On the other hand, the rotorcraft design used is particularly unique. Forgoing the traditional main rotor / tail rotor design, Stanford has opted for a more radical quad rotor design. Essentially, the craft consist of a cross with arms of equal length, with a rotor attached at each tip of the cross. Figure 5 shows a STARMAC rotorcraft [Teo, 2004].

The primary focus of the DragonFly project was to develop a system to prevent two cooperating robots from physically colliding with each other, while maintaining autonomous operation. Given that a vehicle "blunders" into a path of the other, the developed algorithm should allow for the "evader" to fly around the "blunderer" and

Figure 5. STARMAC Rotorcraft [STARMAC, 2008].

continue with its mission. This maneuver is known as an Emergency Escape Maneuver (EEM).

Pursuit-evasion game theory is used to prescribe the best maneuver to escape from a pursuer (blunderer). While a variety of different maneuvers were possible, the DragonFly program chose to limit the EEM to only two choices: when the evader is leading the pursuer, the evader will accelerate, climb, and turn 45-degrees; when the pursuer leads, the evader maintains speed, climbs, and turns 60-degrees. The climb is included merely to gain altitude, not as the main method of avoiding the pursuer.The automatic control system of the DragonFly UAVs is described in detail in [Jang, 2003] as well as in [Teo, 2004]. Both high fidelity nonlinear and linear models of the DragonFly aircraft were developed, as well as control algorithms based on these models.

2.1.6 Elemental Maneuvers

In [Gancet, 2005], the creation of a distributed control framework for the coordination and control of unmanned air vehicles using elemental maneuvers is described. Elemental maneuvers are the abstract, high-level commands given to a single UAV. Team maneuvers are also considered, which abstract team operations using elemental maneuvers and coordination constraints. Within the coordination structure of team operations are vehicle supervisors for each UAV, team supervisors, and the links between them, which define the dynamic information structure and roles of each vehicle. The team supervisors and vehicle supervisors interact with one another through a simple coordination protocol. The distributed control problem is modeled in the framework of dynamic networks of hybrid automata. Dynamic optimization and nonlinear control techniques are used to create elemental maneuver controllers.

2.1.7 STOMP

STOMP [Jones, 2003], Simulation, Tactical Operations, and Mission Planning, is a program put on by U.C. Davis and Lawrence Livermore National Laboratory. The STOMP software architecture is a framework for the simulation, control, and communication of unmanned air vehicles (UAV). Essentially, the system considers itself as a distributed sensor network. One interesting feature of the STOMP architecture is the ability to perform hardware-in-the-loop testing. Thus, real UAV units can provide feedback state information while interacting with the environment. The result is enhanced support for simulation of dynamic and complex events.

STOMP is a powerful tool for the testing of new algorithms involving cooperation, communication, command, or control of a network of UAV units and wireless sensors. The designer can use visual editing software to assemble and configure each simulation down to the state of every object, individually or in groups. In addition, events can be defined and scheduled for specified times during the simulation. However, simulation alone is not enough to completely test and model the dynamics of a UAV system. Thus, hardware-in-the-loop (HIL) capability is built into STOMP. The STOMP virtual environment can then be connected to real UAV units collecting real data as well as simulated data. Figure 6 shows virtual and real UAV communication [Jones, 2003].

The HIL ability of STOMP is what it makes it most interesting in terms of this paper. When real units are used in the system, it becomes less of a simulation and more of an actual ground-station, providing coordination and control to all member UAV units in the system. In [Kent, 2002], STOMP was used to rapid-prototype and test a new cooperation and path-planning algorithm for large UAV networks. Other multi-robot systems could also be rapidly-prototyped on top of the existing STOMP system and easily simulated, then HIL-tested. The result is a flexible ground station that can be easily configured and updated with new functionality along the way.

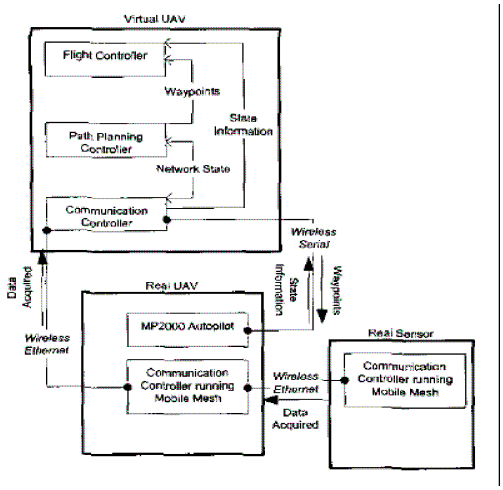


Figure 6. Virtual UAV and Real UAV Communication [Jones, 2003].

2.1.8 DSP-Based Control of Mobile Robots

The Fully Autonomous Advanced Vehicle (FAAK) program [Masar, 2005; Masar, 2004] of the Process Control and Control Engineering Department of the University of Hagen seeks to develop a rapid modeling, simulation, and prototyping system for mobile ground and air robots. The onboard robot low-level control systems are based on programmable DSPs and PID-loops. The vehicles involved include four-wheeled ground vehicles and airships.

Using MATLAB and Simulink, the team designed an integrated environment for rapid prototyping control algorithms based on dynamic models of sensors, actuators, etc. The MATLAB/Simulink system performs the following:

- Allows the creation of an environment for the robot.
- Simulates the interaction of multiple robots in the system
- Generates appropriate code for the onboard DSPs automatically.
- Provides a communication and control interface (central control station) for all the robots.

For the individual robot, the DSPs perform the following:

1. Evaluates sensor data and provide appropriate control signals for the actuators.
2. Navigates the robot throughout its environment by way of reading the onboard odometry (based on encoders).
3. Communicates with the MATLAB/Simulink external monitoring and control system.
4. Also, performs dynamic event handling such as
    - Collision avoidance.
    - Automatically learning system parameters.
    - Generating environment maps.
    - Path planning.

Additionally, the system is equipped with an onboard image processing system for visual servoing. A higher-performance DSP is used for the image processing system and linked to the navigation DSP through dual-port memory. An overall control system diagram is shown below. Figure 7 depicts the FAAK control system [Masar, 2005].

One of the advantages of the FAAK system is the ability to perform hardware-in-the-loop (HIL) simulation of a robot control system. Afterwards, the control system code for the robot can be automatically generated. When testing the generated program on the robot, both online data transfer (real-time wireless communication) and offline data transfer (data logging) are possible to examine the results.

While the FAAK MATLAB/Simulink system can act as a complete multi-robot system with its built-in communication system, its use is primarily for the rapid prototyping of DSP-based control systems for mobile robots. No high-level control possibilities, such as task-planning and scheduling, are yet available within the system.

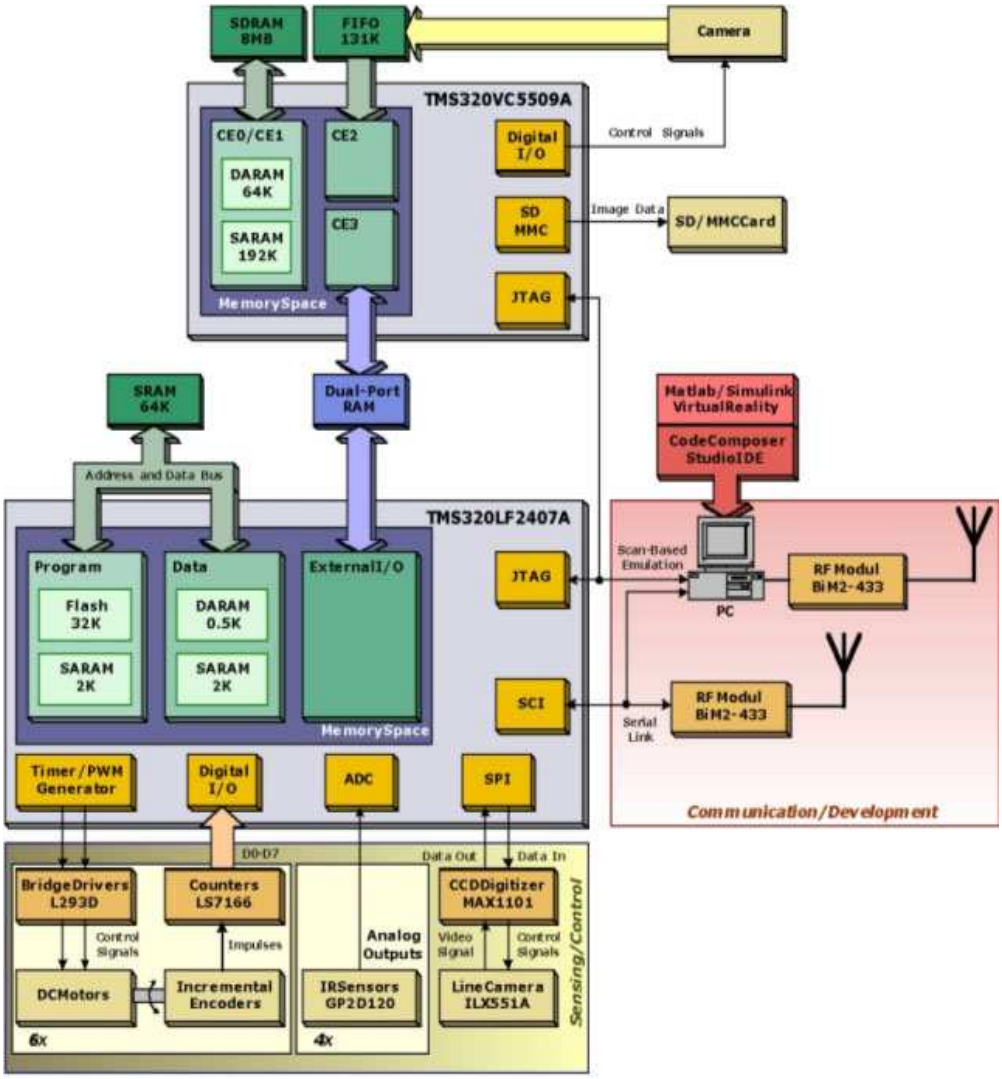


Figure 7. FAAK Control System [Masar, 2005].

Another type of control system used in the field of vehicle control is the intelligent agent. An intelligent agent, or simply agent, typically "wraps" around the native control system of a robot or vehicle, accepting high-level commands from an operator and issuing sequences of low-level commands to the proprietary control system. Additionally, agents have inherent intelligence that allows them to plan autonomously, as well as coordinate and communicate with other agents. Moreover, agents can typically be organized hierarchically to achieve teams and teams of teams.

One of the popular applications of an agent is for the control of hybrid systems. The systems are hybrid in the sense that they have both discrete and continuous states. Several of the programs surveyed below have hybrid control systems at their core.

2.1.9 Agent-Based Mission Management

In [Karim, 2004], the use of intelligent agents to design, implement, and test a Mission Management System (MMS) for a small UAV is proposed. The JACK Intelligent Agents [Agent Oriented Software, 2003] programming language was used for the implementation of the agent-programming paradigm. An autopilot flight control system was already onboard the UAV, providing low-level control. The agent-based system provided a high-level mission-management interface. Thus, the UAV has a higher level autonomy, appropriate to multiple UAV (swarm) situations.

Agents are the central building block of agent-based systems. Akin to objects in the object-oriented software engineering paradigm, they are generally described as computer systems with two important and distinguishing capabilities. Firstly, they are capable of fully autonomous behavior, which entails independent reasoning, decision making, and action in order to satisfy the agents' assigned goals. Secondly, they are situated in an environment which contains other agents with which they can interact by way of social protocols such as coordination, cooperation, negotiation, etc. The JACK agent-based programming language was used. JACK is built in Java, and can be used to build autonomous software systems that are both goal-directed and reactive. The systems that are built with JACK can be any type of distributed reasoning entity and will be able to cooperate through the JACK system.

The developed agent sits at the top layer of the control system, as pictured below. High-level waypoint commands are issued from the MMS, and the autopilot or flight control system simply navigates to the specified waypoint. Figure 8 depicts the Agent—FCS architectural design [Karim, 2004].

The JACK programming language consists of several constructs: agents, capabilities, events, plans, and beliefs. At the highest level of abstraction are agents, which represent entities with autonomous behavior within the system. Capabilities are also abstract entities, but they encompass groups of related events and plans.

Within this system, events are handled by plans. Plans can then post one or more other events. The dynamic selection of plans entails autonomy. Combining all of these constructs yields behaviors centered on the Belief Desire Intention (BDI) theory of agents.

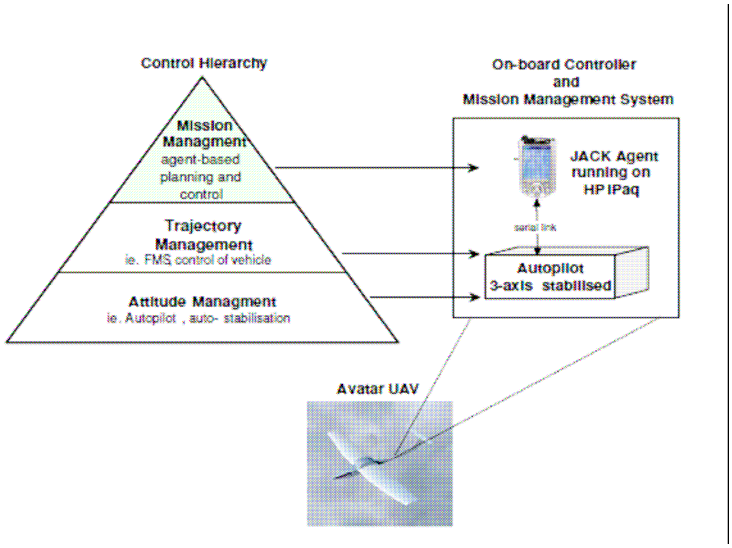


Figure 8. Agent – FCS Architectural Design [Karim, 2004].

BDI is similar to the rational process of the human mind. Each agent focuses on the goals (desires) given to it, making plans (intentions) according to current data (beliefs). The final design of this system is in fact based on the OODA approach, popular in military applications: Observe, Orient, Decide, Act. Essentially, the agent samples or observes its environment, orients itself based on its observations, decides its next course of action, and then acts upon its plans.

While this system was only tested on a single UAV, it could be extended to coordinate multiple UAVs using the same agent-based framework. The paper investigated the use of Contract Nets or Blackboard systems to coordinate multiple units.
A Contract Net protocol uses managers to broadcast a set of tasks to other contractors within the team. The contractors examine the tasks collected and submit a "bid" to the manager agent with a plan to achieve the task or mission. The manager then selects the "bid" it deems most appropriate. A contractual obligation is then made between the manager and contractor, thus producing a "contract net."

Blackboard systems are popular in many current multi-robot systems, such as [Ollero, 2005]. In this system, communication is made by creating a central database that each agent regularly accesses, and creates a local copy of. Thus, message passing is made by writing to the blackboard. Additionally, the state of the world is typically kept in the blackboard for access by all agents. This method allows the incremental update of the model of the world by each member.

Either of the two systems, contract net or blackboard, could be used to extend the agent-based mission management system presented above to coordinate multiple agents in a UAV team.

2.1.10 HICA

The HICA (Hybrid Intelligent Control Architecture) [Rathinam, 2004; Fregene, 2001] develops a systems- and control-oriented intelligent agent framework, decomposed into specific kinds of multi-agent systems. An agent is a system or process that can sense, compute, and act within its own environment in a flexible, autonomous way in order to achieve its objectives. Agents can be reactive, where perception is coupled to action without an internal model of the agent; deliberative, where the system reasons and acts based on the internal model of itself and its environment; or a hybrid, which combines the desirable aspects of both. The hybrid agent is used here. Thus, the system is hybrid in the sense that it includes systems with both continuous-valued and discrete-event dynamic behavior, and also because it allows the use of hybrid agents. Knowledge-based control and coordination can then be integrated with hybrid control primitives, achieving coordinated control of multiple dynamic systems with multiple modes of operation. Figure 9 shows HICA in an environment [Rathinam, 2004].

Planning and coordination within the HICA system is based on partial knowledge-based planning (PKBP). An agent can find a short sequence of modes that to achieve a desired short-term objective while coordinating with other agents to attain team goals. Thus, the agent acts before a complete sequence is generated, letting the PKBP planner perform multiple-passes dynamically.

The HICA can be composed into a multi-agent system by creating a network of agents, a supervisory agent team, or multiple supervisory teams. In the network of agents, the output of each agent is available as an input to all other agents. When the supervisory agent team is added, the global coordination is improved by the utilization of a supervisor agent that receives the coordination output from the network of agents and outputs supervisory events to the network and coordination events to other supervisors. When multiple supervisory teams are used, the teams coordinate using the supervisors to deal with certain aspects of a problem in order to solve the overall problem.

The simulation experiment used in this program is most interesting, as it focuses on the use of both air- and ground-vehicles in a pursuit-evasion war game. In this game, the air vehicles focus on locating a target enemy ground vehicle while the friendly ground vehicles coordinate to perform the actual apprehension of the enemy unit. The simulation concluded with the friend ground vehicles flanking the enemy vehicle, a “capture.”

An application of the HICA [Rathinam, 2004] paradigm is presented in [Fregene, 2004]. In this system, multiple unmanned ground vehicles (UGV) are used for intelligent terrain mapping. Each vehicle is a HICA with an embedded hybrid control system, planning, and coordination logic. All logic required for mapping and terrain traversal is embedded within the individual agent. The team of HICA UGVs coordinates individually and with a supervisor agent, whose responsibility is also to update the centralized map. Decentralized multi-agent control and coordination is the result.

The proposed system is a combination of three research areas: hybrid agent control, path-planning and traversal, and vision-based terrain-mapping. The results show promise for the use of hybrid intelligent control agents in real-world applications.

In [Fregene, 2003], the HICA system was used again to model a multi-vehicle pursuit-evasion game. Two supervisory teams of HICA were implemented to demonstrate control and coordination logic. The scenario was then simulated.

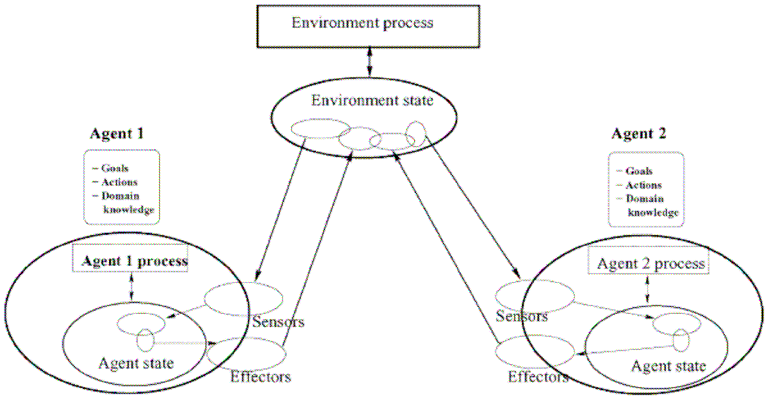


Figure 9. HICA in an environment [Rathinam, 2004].

One supervisory team consisted entirely of air vehicles: one supervisor and three HICA. The second team consisted of ground vehicles: one supervisor and two ground vehicles. Key to this particular paper was the separation into two supervisory teams. When composed in this manner, this class of multiagent systems behaves like a Discrete Event System [Fregene, 2002]. This DES has constrained supervisory output events rates at the team, or macro, level of abstraction. Additionally, this DES framework was used to produce preliminary results at both micro- (individual HICA) and macro- levels [Fregene, 2003].

2.1.11 Framework for Coordinated Control of Multi-Agent Systems

A distributed framework is proposed in [Karray, 2004] for the coordination and control of a multi-agent system. The agents are modeled as Coordinated Hybrid Agents (CHA), which have intelligent coordination control layer (ICCL) and hybrid control layer (HCL). In ICCL, the planning, coordination, decision-making, and computation occurs – equivalent to the high-level intelligence seen in other programs. In the HCL, essentially the native control system of the plant, the control signals are generated for a process according to the commands of the intelligent coordination control layer. The system seeks to create a framework of decentralized control and coordination of its agents.

The target platform for this program is much different from others surveyed. Instead of mobile unmanned vehicles (UV), this system is tested on a system of multiple cranes. While the physical dynamics are surely different, the notion of “intelligent agents” is readily extendable to other platforms, such as air or ground robot vehicles. In this system, the ICCL is able to control the plant in an abstract way, allowing the HCL to handle the actual low-level control signals. The ICCL is built upon the action executor, planning the sequence of discrete events, or coordination states, for the HCL as well as communicating with the supervisor agent and its neighboring agents. A coordination rule base, inspired by social laws [Shoham, 1995], coordinates the actions of the agent by considering the optimal actions and constraints on the agent. The intelligent planner uses the coordination rule base and methods such as potential fields, fuzzy logic, neural

networks, or knowledge-based planning to create the sequence of events for the HCL. Also, the ability to directly communicate with other agents through the communication mechanism is implanted within the ICCL.

The experiment was verified by simulating the coordination of five mobile crane robots. Additionally, two industrial overhead cranes were coordinated to test the system in an environment where the cranes must manipulate payloads in the same workspace without collision.

2.1.12 Hybrid Algorithms of Multi-Agent Control

The multi-agent robot system (MARS) [Timofeev, 1999], created by the St. Petersburg Institute of Informatics and Automation of Russian Academy of Science, coordinates multiple robot-agents in real-time in a dynamic environment. The artificial intelligence is based on neural network control strategies. Also, the multi-agent control system is organized hierarchically with parallel processing features. Both strategic/supervisor and tactical/local levels of control are available within the system.

MARS is similar to COMETS in components, but probably the lesser in sophistication. MARS is a distributed intelligence multi-agent system comprised of executive, information, control, and communication subsystems. The information subsystem encompasses perception, and the executive motion. Information is sotred within open distributed databases, or knowledge bases. Figure 10 demonstrates the MARS architecture [Timofeev, 1999].

Within the multi-agent control system (MACS), the supervisor level performs the following operations:

1. Decomposition of tasks into sub-tasks for the agents.
2. Planning optimized distribution of tasks between agents.
3. Multi-agent global modeling of the environment and dynamic collision avoidance.
4. Conflict resolution between agents to ensure coordination.
5. The operation level performs the following:
6. Local goals and constraints are created.
7. Local environment modeling (obstacle modeling) and agent navigation (path-planning).
8. Optimized scheduling of agents' paths.
9. Programming behavior of the agent by calculating sequences of atomic movement tasks.
10. Agent actuator control.

Real-time path-planning is possible in this system through the use of hybrid artificial intelligence/neural networks. The path-planning system was simulated using the proposed algorithms with a network of twenty-four robot-agents. The simulation results verified that the algorithms provided globally and locally optimal/collision-free paths in a complex multi-agent system.

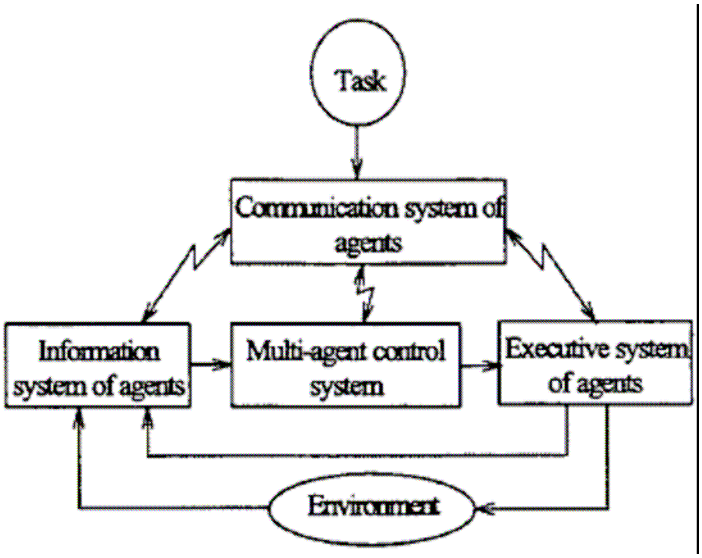


Figure 10. MARS Architecture [Timofeev, 1999].

2.1.13 Intelligent Systems for Autonomous Aircraft

In [Lee, 2004], a mode-driven intelligent agent is developed for the control of autonomous aircraft. A neuro-fuzzy inference engine is implemented to infer the current mode from sensor data, as well as provide abstract input for decision-making processes, which are based on pilot-type aircraft modes. VS/TOL UAVs are the target platform for this project. A block diagram of the overall system is shown below. Figure 11 shows the developmental model of the IFD [Krishnamurthy, 2000].

The primary component of the system is the Intelligent Flight Director (IFD), heavily based on the reasoning and inference methods used by human pilots such as sequences of maneuvers and high-level commands. In the diagram of the IFD above, the Command Logic block translates high-level commands into maneuver sequences. The Trajectory Comparator calculates the appropriate trajectories as maneuvers progress. The Switching Logic selects the most appropriate state trajectory strategy for each maneuver. Finally, the Mission Segment Modifier infers the current maneuver from the sensor data.

As shown, a Knowledge Base is used along with the intelligent agent to provide suitable trajectories for the maneuvers created by the Command Logic block. Four pieces of data are obtained from sensing mechanisms: rates of change of forward velocity, bank angle, heading, and altitude. Six Maneuver Segments (low-level sequence operations) can then be executed depending on the data, including Level Acceleration, Rolling Transient, Steady Turn, Climbing Turn, Aggressive Turn, and Straight and Level. Finally, the available Maneuvers to the system (essentially high-level operations) include En Route Turn, Holding Pattern, Vertical Break, Horizontal Break, and Cruise. Currently, formal testing of the system has not taken place. However, the current research into the IFD has shown promise for a modern VTOL UAV control system.

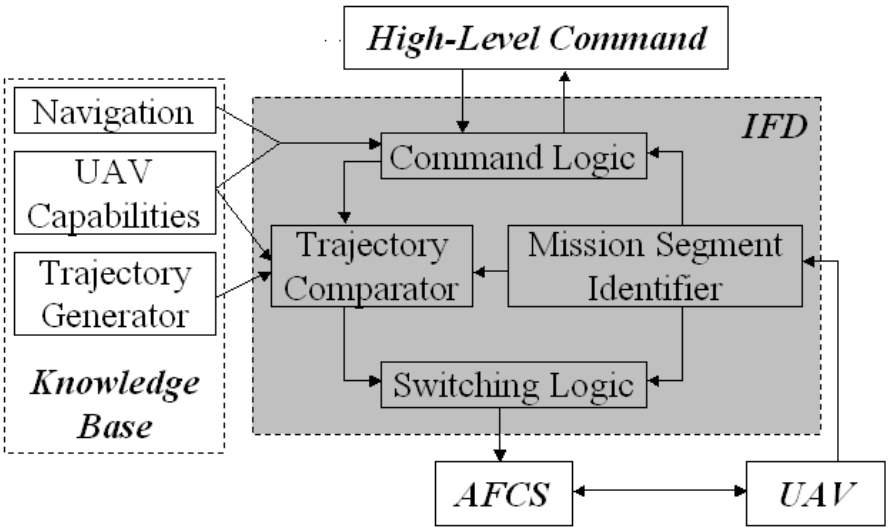


Figure 11. Developmental model of IFD [Krishnamurthy, 2000].

2.2 UAV Flight Control Systems

In addition to surveying multi-robot control systems, significant research was performed into the current literature of single robot low-level flight control and navigation systems for Unmanned Air Vehicles (UAV).

2.2.1 Traditional R/C Systems

Recently, converting standard radio-controlled (R/C) vehicles (miniature cars, planes, helicopters, etc.) to become fully autonomous vehicles has become a popular research trend – both for its low cost and the ability to use commercial-off-the-shelf (COTS) parts. However, most R/C vehicles, even the most advanced, typically use a one-way FM transmitter-to-receiver configuration for communication with the operator. Furthermore, these systems typically have a single fixed-frequency generator (crystal) inside both of the transmitter and receiver. The limitation to a single frequency can generate problems in multi-robot environments. (The FCC has only approved about 30 channels for surface vehicle use in the 75.4- to 76-MHz band. And for air vehicles, only the frequencies in the 72- to 73-MHz band are available).

In order to combat the ill effects of traditional R/C radios, R/C radio manufacturers, Nomadio [Nomadio, 2005] and Spektrum R/C [Horizon Hobby, 2005], independently created custom radio systems based on the popular 2.4-GHz frequency band. Nomadio's product, "Sensor" and Spektrum R/C's product, "DX3," replace standard R/C radios on ground-based miniature-scale race car vehicles. Additionally, both systems utilize digital spread spectrum (DSS) modulation for communication. Finally, both have the ability to frequency-hop around multiple channels to avoid

interference, and both also multicast commands on several different channels to ensure the transmission makes it through to the vehicle.

Nomadio's system, in particular, provides several distinct advantages over standard R/C transmitters-to-receiver pairs:

- Two-way digital communication providing real-time telemetry from the vehicle.
- Can use up to thirty-two available channels for telemetry and control. (Comes standard with three channels of telemetry).
- Four servo channels at a resolution of 4096 steps per channel and a frame rate of 100 Hz (the highest servo resolution currently available in the R/C world).
- Uses Digital Spread Spectrum (DSS) modulation with frequency-hopping on the 2.4-GHz band.
- Eliminates the need for many different fixed-frequency crystals.
- Race information can be saved for later examination with a PC.

The three default telemetry channels are filled with speed, temperature, and battery voltage sensors included with Nomadio's package. Four additional channels are used for high-resolution for servo control. Furthermore, the system has expansion capabilities for up to 32 total channels, allowing the later addition of custom sensors. Extra usability features are also included, such as tactile and audio alerts to telemetry data. Spektrum R/C's system uses a similar radio system to Nomadio's, but does not yet include telemetry functions. Two servo channels are available with a resolution of 4096 steps. Essentially, the current DX3 system is only a solution to channel hijacking with the additional benefits of high-resolution servo control. Currently, Spektrum's website states that real-time telemetry functions will be available Fall 2005 in the form of a plug-in module. Table 2 summarizes the features of these systems.

Table 2. Comparison Table of Modern R/C Transmitters

	Nomadio Sensor	**Spektrum R/C DX3**
Modulation	Digital Spread Spectrum 2.4 GHz	Digital Spread Spectrum 2.4 GHz
Radio Model	Direct Sequence Spread Spectrum	Direct Sequence Spread Spectrum
Range	1000 ft (LoS)	3000 ft. (LoS)
Frame Rate	100 Hz	Unknown
Latency	5 -10 ms	5.6 ms
Servo Channels	4	2
Channel Resolution	4096 steps per channel	4096 steps per channel
Telemetry Channels	3	0 (currently)

2.2.2 Current UAV Helicopter Autonomous Systems

In [Lee, 2004], an autonomous flight, navigation, and guidance system was developed that utilized a ground station for telemetry and command. For this project, an airship was chosen for its natural characteristics of long-duration flight and gradual degradation under system failures. The flight system implemented here controls all flight operations of the airship using an autopilot system and control laws based on the dynamics of the vehicle. Sensor information is returned to the ground station during flight using the onboard communication system, and commands are received from the ground station. Finally, all actuator and status subsystems are maintained by the flight control package.

The COMETS system has performed significant research into helicopter UAV control systems. In [Gonzalez, 2004], research entailed control and stability analysis of autonomous helicopters. Utilizing both linear and non-linear control laws, a two-time scale decomposition of helicopter dynamics was created. The fast subsystem deals with the rotational dynamics, and the slow subsystem with the translational dynamics. A Lyapunov function models the stability of the fast dynamics. Additionally, feedback linearization stabilizes the slow dynamics. Using these two subsystems, a linear control law was created and analyzed. Finding significant drawbacks to the linear system, non-linear control laws are also developed and presented within [Gonzalez, 2004]. A block diagram of the overall system is shown in Figure 12 [Gonzalez, 2004].

Several papers surveyed examined low-cost flight control systems for autonomous helicopters. In [Roberts, 2003] a system is implemented that does not rely on traditional helicopter UAV sensing means, Inertial Measurement Units (IMU) and Global Positioning Units (GPS). Instead, a low-cost inertial sensing package (developed in-house) and a pair of CMOS cameras are used for navigation. The advantage of this method is the inclusion of integrated stereo vision system for further expansion into the realm of 3D visual servoing techniques as well as significantly less per-unit costs.

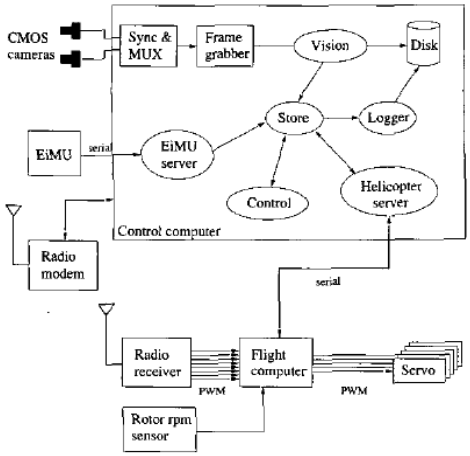


Figure 12. COMETS overall system setup [Gonzalez, 2004].

Vision-based systems were also utilized in [Sinopoli, 2001] and [Kanade, 2004]. In [Sinopoli, 2001], also performed at U.C. Berkeley, each UAV is equipped with a GPS/INS (Inertial Navigation System) package as well as onboard cameras. Low altitude navigation is the goal. Real-time creation of a 3D model of the world is performed in-flight, creating a partially-known 3D environment for the UAV to navigate around using path-planning techniques. Image sequences are processed using multi-resolution wavelet transforms and fused with sensor data. Using the collected data, a path is planned from a start point to an end point, refining and updating the path in real-time as it executes. The target platform for this technology is the BEAR helicopter UAV found at [Berkeley Aerobot, 2005].

The visual techniques used in [Kanade, 2004] focused on recovering motion and structure from a video sequence, as well as 3D terrain mapping from a laser range finder. Both of these sets of data are fused with an onboard GPS/INS package. The system is targeted at small and micro helicopter UAVs. By using 3D vision techniques, positional information can be obtained relative to objects within the environment. Path-planning can then ensue based on the created local world model. Simulations have shown promise for further research into 3D visual navigation techniques by comparing GPS/INS-based state estimation versus the vision-based estimator, or the visual-odometer. An overview of the vision system is shown in Figure 13 [Kanade, 2004].

Systems based entirely on GPS/INS data are also prevalent in current research literature [Yoo, 2003; Kahn, 2003; Saphyroon, 2004; Sasiadek, 2004; Accardo, 2004]. A navigation system based solely on GPS is described in [Hui, 1998].

The sensor fusion of these two devices, GPS/INS, is examined in [Sasiadek, 2004], particularly focusing on proper utilization of multiple GPS satellites. The fusion system is based on a Kalman Filter, and the satellite selection algorithm on a Dilution of Precision (DOP) technique. Simulation experiments were performed for both real-time error state feedforward and feedback Kalman filters with the selection of up to four satellites simultaneously.

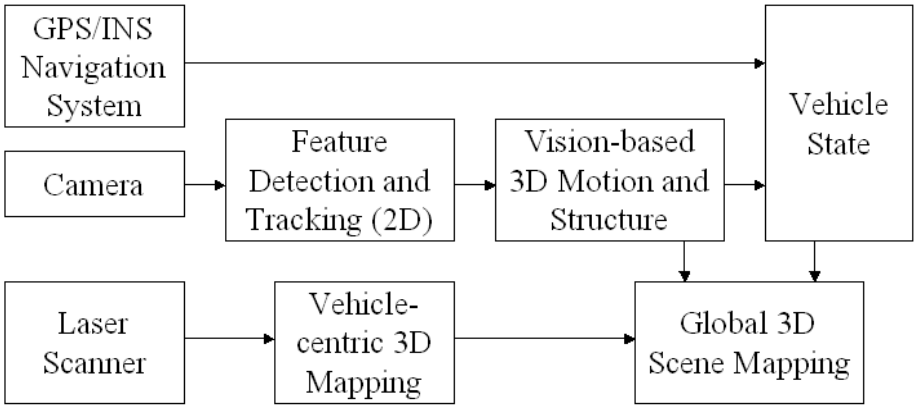


Figure 13. Overview of the vision system [Kanade, 2004].

A low-cost GPS/INS-based UAV navigation system is presented in [Yoo, 2003], but targeted at fixed-wing UAVs rather than helicopters. Strict design constraints were placed on the sensing system, including very low weight, small dimensions, low power, reasonable accuracy, fast update rate, and finally a fast data rate. Aside from the common GPS/INS sensing system used in other projects, this particular system investigated the use of multiple GPS antennas for boosting signal strength and avoiding complete loss of satellite signals. The sensing and fusion subsystem is pictured below. Field tests of the sensing system were performed on the ground using multiple antennas, verifying the initial constraints on the system. Flight tests will ensue in the future. Figure 14 depicts the structure of the GPS/INS fusion system [Yoo, 2003].

Another low-cost UAV controller is presented in [Saphyroon, 2004]. In this particular system, the constraints call for an embedded microcontroller system and COTS hardware and software. An in-house constructed sensing package was created, including a Pitot-static tube (collects airspeed, altitude, and angle of attack), absolute pressure sensor (can be used to obtain altitude), tilt sensor (measures Euler angles), rate gyroscope (measure rate of change of the Euler angles), accelerometers, a digital compass, and GPS. Only initial testing of the sensing system has taken place to date. The main advantage of this system is it's very low cost per unit along with fast real-time operation due to the embedded microcontroller.

In [Buskey, 2001], an INS and an artificial neural network (ANN) are combined to perform autonomous helicopter hovering. One advantage of this system is an ability to deal with sensor errors and maintaining correct operation. The system is purely reactive to the environment, using only INS data. Future research will include a stereo-vision system for a higher level of accuracy in low-altitude hovering. Research developed in [Nakanishi, 2004] also pursues the use of neural networks in UAV control, but also considers stochastic uncertainties, such as wind direction and speed – both of which have a significant effect on helicopter flight dynamics.

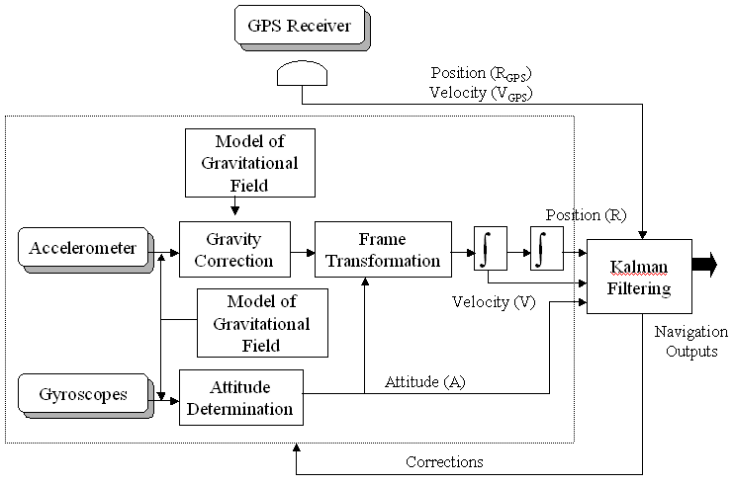


Figure 14. Structure of GPS/INS Fusion System [Yoo, 2003].

A similar experiment to those described above was performed in [Kahn, 2003]. Miniature R/C helicopters are very unstable and require mechanical dampers, or flybars, to assist the human pilot during flight. Unfortunately, the flybar also greatly increases the complexity of the helicopter rotor dynamics when attempting to model the system, as well as increases the drag profile of the helicopter. The project described in [Kahn, 20039] attempts to circumvent the problems presented by the addition of a flybar by removing it from the system entirely. Instead, a lagged-rate feedback system or an attitude-command attitude- hold (ACAH) system, based on adaptive neural network model inversion control, could be used to assist the pilot during flight. Flight tests and simulations of the lagged-rate system are presented within the paper. While the system is not a "true" flight control system, it provides limited pilot assistance and has the potential to be extended to a full autopilot.

Only a handful of the vast amount of research into UAV flight control systems has presented here. However, the research trends begin to become clear by looking at these individual systems. First, GPS/INS sensor fusion is the primary means of sensing the environment, with some departures to only-GPS or only-INS (or IMU) systems. Several interesting vision-based systems are on the horizon. Most vision systems are based on stereo-vision and 3D modeling of the environment, while others include laser-range systems. Many of the systems are built specifically for airships, fixed-wing aircraft, or helicopters. However, most systems are applicable to *any* aircraft.

2.2.3 Brief Survey of Commercial Miniature UAV Autopilots

An autopilot system for an R/C helicopter can perform a variety of operations, from hovering the vehicle in position without constant user input, to navigating along a path of GPS waypoints. Most also allow integrated sensor data (INS, GPS, etc.) to be logged and/or transmitted to a ground station. Some advanced autopilots also include user-configurable loops for extra control over camera stabilization, etc. The purpose of this report is to analyze the many available commercial autopilots to find a best-fit for the IRIS lab's electric helicopter.

The following autopilots are considered:

1. MicroPilot mp2028g with HORIZON ground station and XTENDER development kit ($5,500 autopilot + $5,000 software development kit).
2. CloudCap Technology Piccolo II system kit ($50,000 for airframe modeling and hardware)
3. weControl wePilot1000 flight control system ($35,000 to $45,000 for airframe modeling and hardware)
4. Procerus Kestrel Autopilot 2.22, developed at BYU MAGICC lab ($5,000 autopilot + $2,995 ground station and software kit)
5. Rotomotion UAV Flight System Helicopter Controller ($5,500 autopilot and software kit)

The final recommendation was the MicroPilot mp2128g. The runner-up was the Procerus Kestrel.

First, the mp2028g was originally designed for fixed-wing applications, but recently provisions have been added into the code to allow the autopilot to hold a helicopter in a hover. This autopilot includes a complete ground station software package (HORIZON), and has several advanced features beyond basic assisted manual control and waypoint navigation. The telemetry can be customized to provide user-defined fields, video overlays can be configured, and user-defined loops can also be created for control over camera stabilization, or other payload devices. The system can be made fully autonomous, from take-off to landing.

Additionally, the mp2028g can perform pre-programmed flight maneuvers, such as changing altitude and flight speed at each waypoint. The software development kit allows the creation of custom control laws, communication, ground control software, custom payload control and data collection, and access to the complete autopilot state. While the most expensive solution, it is also the most complete and expandable.

Secondly, the Piccolo II was created specifically with helicopters in mind, based on the previous Piccolo Plus system. Like the mp2028g, the system can be made fully autonomous, from take-off to landing. The system also has graceful degradation features that allow it to cope without GPS data. Assisted manual mode and waypoint navigation features are included with the Piccolo II and ground station software. The communications SDK is also included for custom interfaces. The main drawback is the lack of custom programmability such as that found in the mp2028g.

Unfortunately, the cost of CloudCap initially modeling the helicopter dynamics and configuring the autopilot (estimated at $50,000) far outweighs the cost of the hardware. While the features are right, the price is not.

Thirdly, the wePilot1000 is a more basic autopilot designed by Swiss company weControl. However, it includes features such as automatic take-off and landing, waypoint navigation, and assisted manual modes. The ground station software is a bit more basic than other competitors, but it provides essential mission planning and status monitoring. Also, a software SDK is not mentioned on the website or brochure, so custom communications and control software might not be possible.
Again, the wePilot1000 will require initial helicopter modeling by the manufacturer, resulting in too great of an initial investment ($35,000 to $45,000). Not recommended.

Fourthly, The Kestrel autopilot originated at the Multiple Agent Intelligent Coordination and Control Laboratory (MAGICC) at Brigham Young University (BYU). The university research led to the production of this commercial autopilot. The Kestrel autopilot is on par with the MicroPilot mp2028g in features, including autonomous take-off, flight, and landing; assisted manual flight modes; a Virtual Cockpit ground station kit; hardware-in-the-loop configuration software to reduce time-to-flight; and a communications SDK to allow for custom 3rd party communication software. The main drawback: helicopter support will not be built into the Kestrel until late 2006, early 2007. The main advantage, aside from the extensive features: previous university research has provided a bevy of published papers about the autopilot and software development.

Lastly, The Rotomotion autopilot is not as pretty to look at as the previous candidates, nor does it have as many features, but it does provide one distinct advantage: the original autopilot software was written under an open source license, so all source code should be freely (as in beer) attainable and modifiable. However, the initial setup

seems more complicated than the other candidates, requiring Rotomotion personnel to assist, and the hardware is still quite expensive at $5,500. Thus, the author does not recommend this autopilot for a final purchase, but it is an interesting candidate for consideration.

In conclusion, the two most advanced solutions surveyed are the MicroPilot mp2028g and the Procerus Kestrel autopilot. However, neither was has native support for helicopter control. The mp2028g was only recently configured for helicopter hovering at the request of a previous customer, and the Kestrel will not have support till the end of the year. However, if they *do* gain full support in the near future, they will be the most flexible and useful solutions.

The second choice would be the CloudCap Piccolo II, designed to support helicopters. The software package is not as complete as the previous solutions, but it includes the most needed autopilot features with support for custom communication software. Unfortunately, the price is not reasonable for our project.

Finally, the most basic autopilot, the weControl wePilot1000, also provides the most needed autopilot features, but does not include a software development kit. The ground station also seems less advanced than the other solutions, and no hardware-in-the-loop simulation seems available (increasing the difficulty of tuning the autopilot for our helicopter). Again, the price is not reasonable for our project. Figure 15 depicts all five autopilots. Table 3 summarizes the autopilot features.

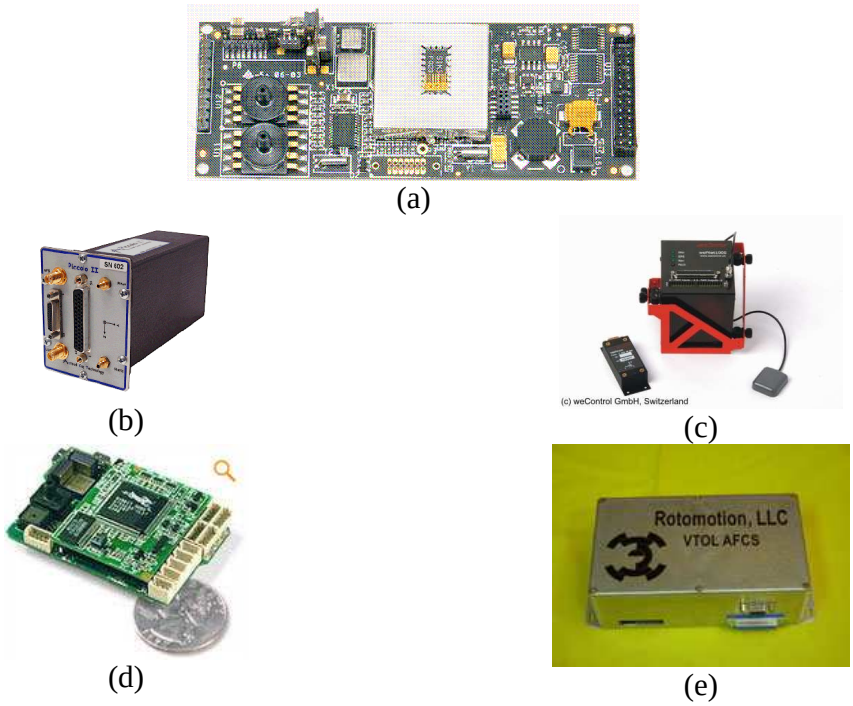


Figure 15. Autopilots: (a) MicroPilot 2128g, (b) CloudCap Piccolo II, (c) weControl wePilot1000, (d) Procerus Kestrel, and (e) Rotomotion UAV Helicopter Controller.

Table 3. Autopilot Features Comparison

Manufacturer	MicroPilot	CloudCap Technology	weControl	Procerus/BYU	Rotomotion
Location	Manitoba, CAN	Oregon, USA	Zurich, SUI	Utah, USA	South Carolina, USA
Autopilot	mp2028g	Picollo II	wePilot1000	Kestrel 2.22	UAV Heli. Controller
Autopilot Cost	$5,500	$?	$?	$5,000	$5,500
Software Development Cost	$5,000	$?	$?	$2,995	N/A
Manufacturer Modeling	$0	<$50,000	$35,000-$45,000	$0	$?
Fixed-Wing Control	Yes	Yes	?	Yes	?
Helicopter Control	?	Yes	Yes	2006Q4	Yes
Autonomous Take-off/Landing	Yes	?/Yes	Yes	Yes	No
Integrated GPS	Yes (1 Hz)	Yes (4 Hz)	Yes	Yes	Yes
Inertial Sensors (Accel./Gyro)	Yes	Yes	Yes	Yes	Yes
Other Sensors	Airspeed, Altimeter	Magnetometer, Altimeter, Sat. Comm.	Magnetometer, Barometer	Wind, Magnetometer	Magnetometer
Telemetry	Yes, with video support	Can be implemented with communications SDK	Yes, using ground station software	Yes, with video support	Yes, optional
Onboard Data Log	Yes (1.5MB)	Yes	Yes	Yes	Yes
Hardware-in-the-Loop Simulation	Yes	Yes	?	Yes	?
Waypoint Navigation Mode	Yes	Yes	Yes	Yes	Yes
Assisted Manual Mode	Yes	Yes	Yes	Yes	Yes

2.3 Sensor Placement Planning

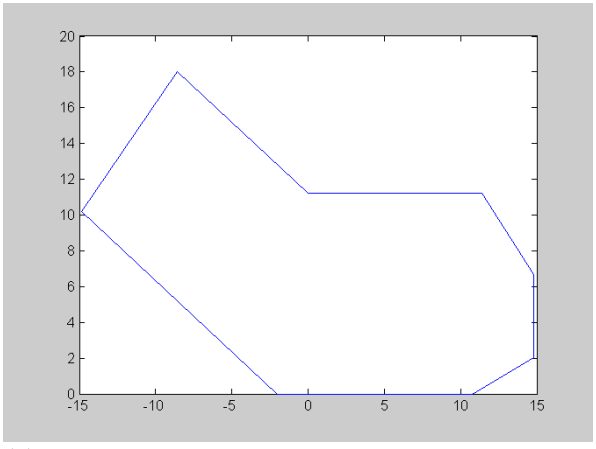
The goal of complete wide-area surveillance using multiple cameras and a UAV helicopter interceptor requires pre-planning to position assets most effectively. Thus, the field of sensor placement planning was investigated for maximizing the visible coverage of an area while minimizing the overall system cost.

2.3.1 The Art Gallery Problem

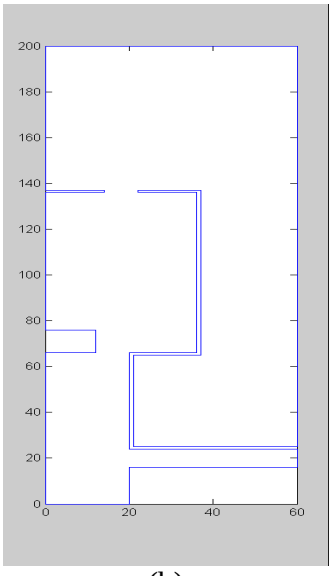
The "Art Gallery problem" is a textbook approach to the camera placement component of an automated surveillance system. The 2-D layout of the outdoor area to be monitored is represented by a simple polygon, *P*, with *n* vertices. Figure 16 depicts two polygonal representations of floor-plan layouts used during experimentation. Figure 16 (a) is Room 209 of Ferris Hall on campus, and Figure 16 (b) is our secondary warehouse lab known as "IRIS West." Both rooms have been simplified in order to create realistic environments that a robot could navigate safely.

For placing cameras within these areas, one of the traditional Art Gallery solutions was used. [O'Rourke, 1987] [Berg et al., 2000]. Figure 16 depicts the simply polygon representations of floorplans.

1. Partition the polygon into monotone sub-polygons,
2. Triangulate each of the monotone polygons by adding diagonals,
3. Three-color the triangulated polygon, and
4. Place guards (cameras) at vertices with least-used color.



(a)



(b)

Figure 16. Simple polygon representations of floorplans (a) Room 209, and (b) IRIS West (simple).

The first step, partitioning into monotone sub-polygons, is significantly simpler, O(nlogn), than the more desirable alternative of partitioning into convex polygons [1]. For large polygons, monotone partitioning is much more practical. The results are typically *y*-monotone due to the sweep line partitioning method, where an imaginary line is swept downwards across the polygon and each encountered vertex is processed along the way. There are typically five types of encountered vertices: start, end, split, merge, and regular. [O'Rourke, 1987] [Berg et al., 2000].

Start and split vertices lie above their immediate neighbors in internal angles of less than and greater than 180º, respectively. End and merge vertices lie below their immediate neighbors with internal angles less than and greater than 180º, respectively. Regular vertices don't fall into any of these categories.

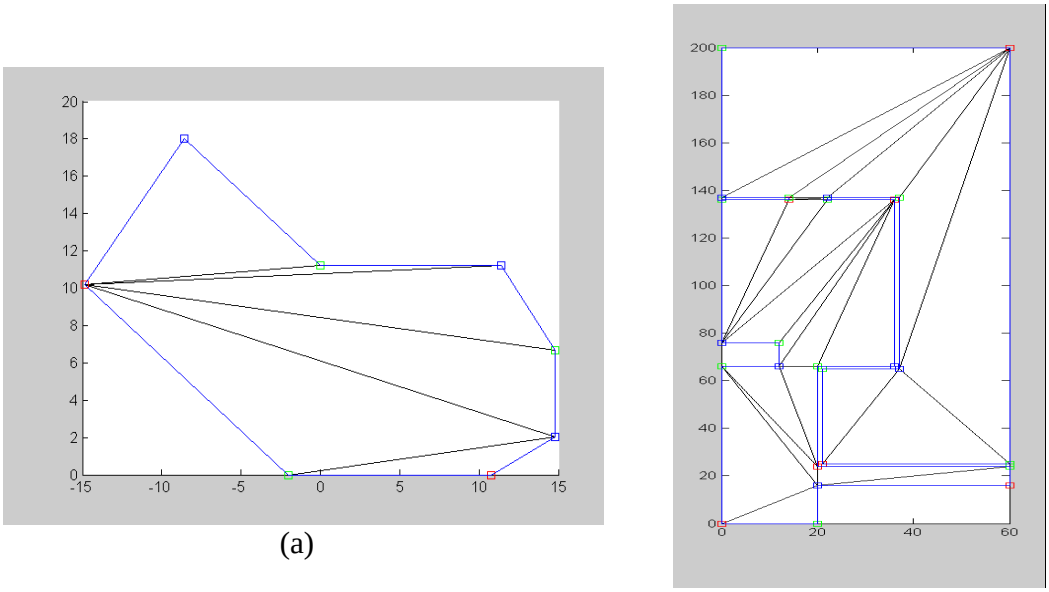
As the sweep line descends, each vertex is processed once and diagonals are drawn to remove split and merge vertices according to the algorithm. Once all split and merge vertices are removed, the polygon will be partitioned into monotone sub-polygons. Please see the references for the detailed algorithm explanation, which is beyond the scope of this paper. Afterwards, in step two, diagonals are added in linear time within the monotone sub-polygons to create a full triangulation.

Implementation of steps one and two was accomplished via a freely-available sweep line polygon triangulation program. This modified implementation is able to handle most polygons with or without holes and output the result to a text file in several different formats. The results can then be parsed easily into Matlab.

Afterwards, step three requires three-coloring the triangulated polygon. For our implementation, we used Kooshesh and Moret's linear-time algorithm [Kooshesh, 1992]. Given a triangulation *T* of a polygon *P* with a perimeter-ordered list of its *N* vertices, p_0, p_1, …, p_n, the three-coloring can be computed based on the order of each vertex. Since the output of polygon triangulation, *T*, is typically formatted in such a way that the number of connections at each vertex is easily obtained, three-coloring can be done in linear time with the following algorithm.

$\text{Color}(p_0) \leftarrow 1$
$\text{Color}(p_1) \leftarrow 2$
for i = 1 to N-1 do
  if odd(deg(p_i))
    then $\text{Color}(p_{i+1}) \leftarrow \text{Color}(p_{i-1})$
    else $\text{Color}(p_{i+1}) \leftarrow 6 - \text{Color}(p_{i-1}) - \text{Color}(p_i)$
endfor.

Placing cameras at each of the vertices with the least-used-color completes step four of the algorithm. For example, Fig. 6(a) has been triangulated and each of the vertices assigned a color (red, green, or blue) according to Kooshesh and Moret's algorithm. Three vertices are colored blue, three green, and two red. Since the least-used-color is red, guards (cameras) should be placed at the red vertices. Figure 17 depicts the triangulated and three-colored test areas.



(a) (b)

Figure 17. Triangulated and three-colored polygons: (a) Room 209, and (b) IRIS West (simple).

Unfortunately, the Art Gallery solutions make several unrealistic assumptions about the guards: 360º field of view and infinite viewing range and focus. While the method is a great start on solving the camera placement problem, a better model of the individual cameras themselves is needed. Thus, the Art Gallery problems and solutions were set aside and different methods were investigated.

2.3.2 Visibility Graphs and Polygons

Visibility deals with finding the visibility polygon from the perspective of a single node within the enclosed area. This polygon encompasses all visible points (i.e., within line of sight) from that node. Calculating this polygon can be done in a brute-force, exhaustive manner by testing the intersection of all edges against each other. Naturally, this leads to a very slow implementation for large, detailed areas. See Figure 18 for an example of visibility polygons.

Assume the polygon, P, has n vertices, v_i, and n edges, e_i, both ordered counter-clockwise around the perimeter. An important part of camera placement is calculating the visibility, or the set of all visible points, from some point, x, within the polygon. Typically, the visibility from a point is represented as a sub-polygon of P. After the visibility is obtained for all valid camera placements, an overall coverage map can be calculated and the best placements chosen.

We consider two types of visibility: vertex and full. The result of vertex visibility is simply a weighted, connected graph depicting all vertices that can "see" each other and therefore have Euclidean distance. Instead of a sub-polygon representing the visibility, we only have an nxn Boolean matrix, VIS, which is true if vertex v_i is directly visible from vertex v_j, and vice-versa. Thus, VIS(j,i) is also true if the pair is directly visible to

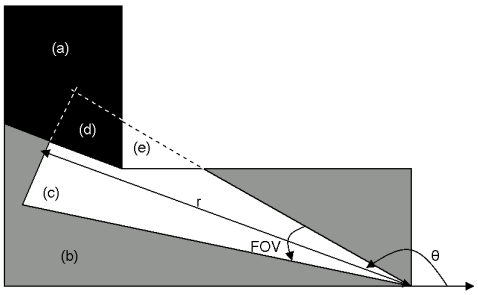


Figure 18. (a) Area that is not visible, (b) Visibility polygon, (c) Camera view, (d) Occluded camera view, and (e) Out of polygon camera view.

each other. This type of visibility is especially useful for shortest-path operations, such as Dijkstra's algorithm [Dijkstra, 1959].

The result of full visibility is a sub-polygon of *P* consisting of all visible line segments from some point *x* within the polygon. The line segments consist entirely of points on the perimeter of *P*. Once calculated, we can use an estimated camera model consisting of pan angle, field of view angle, depth of field, and spatial resolution to simulate the actual coverage of that camera. Based on these parameters, we can calculate a coverage arc (*FOV, r, θ)* for the camera with radius, *r*, field of view, *FOV*, and pan angle, *θ*. Afterwards, a set intersection of the coverage arc with the visibility polygon will give us a good estimate of the actual camera coverage. Again, see Figure 18 for a visual model of this process.

There exists a naïve approach to solving the visibility polygon problem. Essentially, for each vertex v_i, $i = 1, 2, \ldots, n$, we draw an imaginary edge, e_{ij}, to all other vertices v_j, $j = 1, 2, \ldots, n$, $j \neq i$, and test for intersection with all edges not adjacent to v_i or v_j. The overall worst-case complexity is $O(n^3)$. However, if we find an intersection, we can immediately break the testing of edges, declare the pair not visible, and move to the next edge, $e_{i,j+1}$. Additionally, we can also simplify the algorithm by realizing that visibility between vertices *i* and *j* implies visibility between vertices *j* and *i*, or VIS(i, j) = VIS(j, i). Therefore, we only test the imaginary edges, e_{ij}, between v_i, $i = 1, 2, \ldots, n$, and v_j, $j = i+1, i+2, \ldots, n$. The full algorithm is shown below.

While fine for small polygons with few vertices, the $O(n^3)$ complexity can become troublesome as the polygons gain more vertices. A faster $O(n^2 \log n)$ textbook alternative is presented in Berg, et al. [2000] as the algorithm VisibilityGraph(S), but not implemented here. Additionally, the visibility graph must be calculated for individual points on the interior or boundary of the polygon. To accomplish this task, a modification of the previous naïve algorithm was implemented. See algorithm VIS = VertexVisibility(p, v, n).

The notion of vertex visibility will be revisited in further sections when implementing complete visibility polygons and finding shortest-paths. Next, however, an existing complete visibility algorithm is examined.

Algorithm VIS = VertexVisibility(v, n)	
Input	x- and y-coordinates of the n vertices, v_i, i = 1, 2, …, n, of polygon, P.
Output	n x n visibility matrix, VIS, where VIS(i,j) is true if v_i is visible from v_j.

```
for i = 1 to n
   VIS(i, i) ← true
   for j = i+1 to n
      visEdge ← edge(v(i), v(j))
      test ← false
      for k = 1 to n
         testEdge ← edge(v(k), v(next(k)))
         if intersect(visEdge, testEdge)
            test ← true
            break
         endif
      endfor
      if not(test)
         VIS(i, j) ← true
         VIS(j, i) ← true
      endif
   endfor
endfor
return VIS
```

Algorithm VIS = VertexVisibility(p, v, n)	
Input	Point, p, and the x- and y-coordinates of the n vertices, v_i, i = 1, 2, …, n, of polygon, P.
Output	Length n visibility vector, VIS, where VIS(i) is true if v_i is visible from p.

```
for i = 1 to n
   visEdge ← edge(p, v(i))
   test ← false
   for j = 1 to n
      testEdge ← edge(v(k), v(next(k)))
      if intersect(visEdge, testEdge)
         test ← true
         break
      endif
   endfor
   if not(test)
      VIS(i) ← true
   endif
endfor
return VIS
```

2.3.3 Erdem and Sclaroff's Radial Sweep for Visibility Polygons

Testing for intersection with each edge needlessly increases the complexity of the algorithm. Therefore, Erdem and Sclaroff's O(nlogn) Radial Sweep algorithm [Erdem, 2006], an extension of the vertex visibility algorithms by Berg, et al., was implemented. The algorithm sweeps an imaginary ray radially around the point *x* and checks for intersections at each vertex it encounters. The end result is a polygon constructed entirely of visible line segments.

First, the Radial Sweep algorithm converts the edge list to polar coordinates relative to the point, *x*. Secondly, any edges that are intersected at $\theta = 0$ by *x* are split into two separate edges. The start and end vertices of the split edge are the start vertex of the first edge and the end vertex of the second edge, respectively. The end vertex of the first edge is the intersection at $\theta = 2\pi$. The start vertex of the second edge is the same intersection at $\theta = 0$. These edges allow for easier processing during the radial sweep phase, since we won't deal with zero-crossings. The RadialSweep algorithm is shown in Figure 19.

Next, backwards-facing edges (where the end vertex angle of the edge is less than the start vertex angle, $\theta_e < \theta_s$) are pruned from the edge list. Backwards-facing edges cannot be seen from *x* and are removed to save processing time later. We can then construct the list, *Q*, with an entry for each endpoint in the edge list in the form (θ_i, r_i, ε_i), where θ_i is the polar angle, r_i is the radius, and ε_i is the incident edge of the endpoint. If not attached to a backwards-facing edge, each vertex will be represented twice – first as the end vertex of the previous edge and second as the start vertex of the next edge. Thus, the list Q can have up to $2n$ entries. After construction, the list Q is sorted in lexicographically increasing order of polar angle and then radius (end vertex entries are considered less than start vertex entries). . Figure 20 depicts the process [Erdem, 2006].

With the initial data structure creation complete, the algorithm can proceed with the radial sweep. Using the ordered list, *Q*, each vertex is handled based on being a start or end vertex, and the type of intersections possible. Type 1 intersections project the "shadow" of an end vertex onto the edge behind. Type 2 intersections occur at an edge closer to *x* than the target start vertex (usually when that vertex is the end of a backwards-facing edge). Finally, Type 3 intersections project the "shadow" of a start vertex onto the edge behind.

The **insert** function used during Type 2 processing manages the insertion of candidate edges into the sorted list by their distance from *x* (ascending). Event point type cases can be seen in Figure 21 [Erdem, 2006]. Additionally, if the top three vertices of the visibility polygon are ever found to be collinear (which can happen with Type 2 intersections or the pseudo-vertices created by intersecting *x* at $\theta = 0$), the superfluous middle point is removed. Once all vertices in the list, *Q*, have been processed, the complete visibility polygon for point, *x*, has been found. Pseudo-code for the Radial Sweep algorithm is presented below [Erdem, 2006].

Algorithm 1: FindVisibilityPolygon(**ELC**,x)

input : Edge list **ELC** and a point x inside the polygon.

output: **PV**, the list of vertices of the visibility polygon.

begin

- Convert **ELC** to its polar coordinates, **ELP** ;
- Prune all backward facing edges;
- Construct the edge list **Q** ;
- Sort **Q** in lexicographically ascending order;
- **SL** ← NULL;
- **PV** ← NULL;
- CurrentVertex ← **pop** (**Q**);
- ActiveEdge ← **incidentedgeof** (CurrentVertex);
- **push** (CurrentVertex,**PV**);
- **while Q** ≠ NULL **do**
  - CurrentVertex ← **pop** (**Q**);
  - `HandleEventPoint` (CurrentVertex,ActiveEdge,**PV**,**SL**);
  - 1.1 **while** *The top three vertices in* **PV** *are collinear* **do**
    - **delete** (The 2nd vertex in **PV**);
  - **endw**
- **endw**

end

Function `HandleEventPoint`(*CurrentVertex,ActiveEdge,**PV**,**SL***)

input : The current vertex CurrentVertex, the active edge ActiveEdge, the visibility polygon **PV** and the sorted list **SL**.

output: The function operates on the input parameters.

begin

- TYPE1 if *(CurrentVertex is an end vertex) AND (**incidentedgeof** (CurrentVertex) == ActiveEdge)* then
  - **push** (CurrentVertex,**PV**);
  - **while SL** ≠ NULL **do**
    - e ← **head** (**SL**);
    - if ***endvertexof*** *(e) > CurrentVertex* then
      - k ← **intersect** (e, CurrentVertex);
      - **push** (k,**PV**);
      - ActiveEdge ← e;
      - **break**;
    - **endif**
    - **pop** (**SL**);
  - **endw**
- **endif**
- if *CurrentVertex is a start vertex* then
  - k ← **intersect** (ActiveEdge, CurrentVertex);
  - e ← **incidentedgeof** (CurrentVertex);
  - TYPE2 if ***radiusof*** *(k) < **radiusof** (CurrentVertex)* then
    - 2.2 **insert** (e,**SL**);
  - **endif**
  - TYPE3 if ***radiusof*** *(k) ≥ **radiusof** (CurrentVertex)* then
    - **push** (k,**PV**);
    - **push** (CurrentVertex,**PV**);
    - ActiveEdge ← e;
  - **endif**
- **endif**

end

19

Figure 19. RadialSweep Algorithms: (a) Finding the visibility polygons and (b) Handling event points [Erdem & Sclaroff].

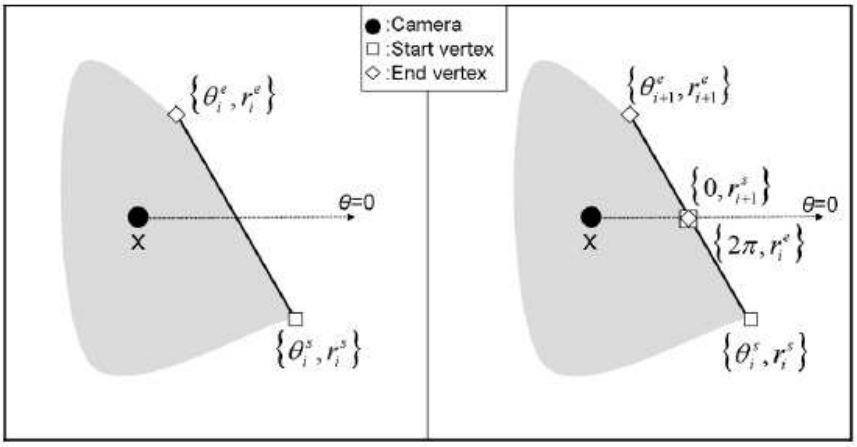


Figure 20. Illustration of splitting the edge i crossing at θ = 0 [Erdem & Sclaroff].

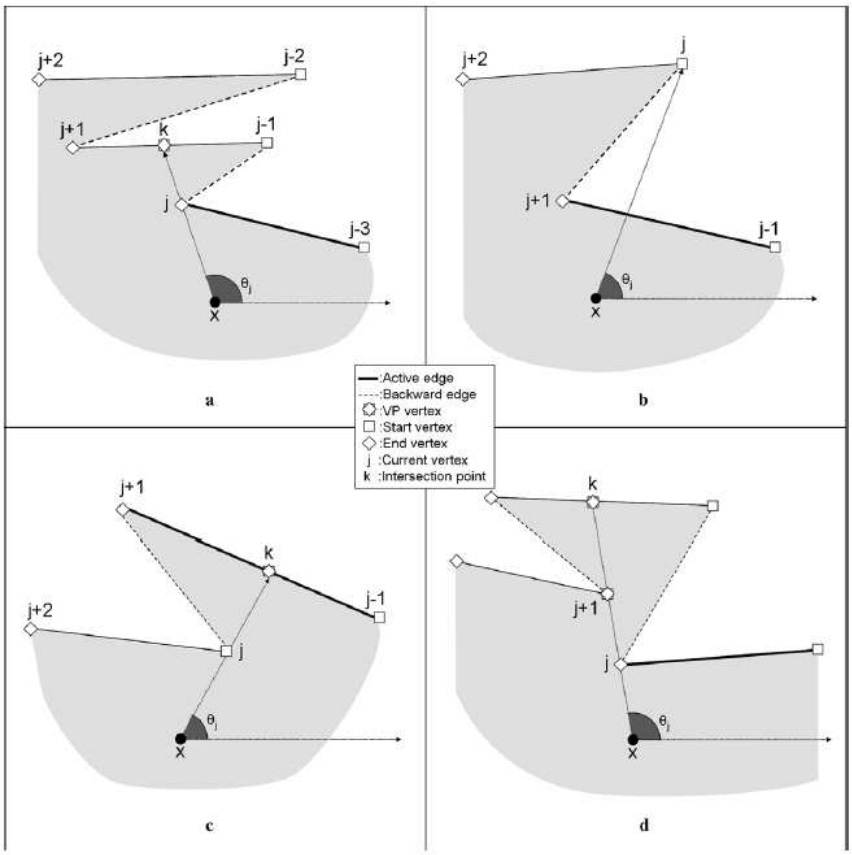


Figure 21. Event point types and a special case, (a) Type 1, (b) Type 2, (c) Type 3, (d) Special case [Erdem & Sclaroff].

3. UAV Helicopter Control System

The overall application of this project is towards automatic, persistent wide-area surveillance. The complete envisioned solution would include optimally placed video cameras using networked automatic intruder detection and localization software, remote ground control system (GCS) software, and an interceptor/tracker UAV helicopter system. Previous work by this author has focused on finding the optimal camera placements. Here, the focus will be on the design of a complete R/C helicopter control system for the interceptor/tracker UAV helicopter, including a central ground control system software package.

The system block diagram for the complete persistent surveillance system can be seen in Figure 22. The face detection and localization presented later in this chapter yields the estimated location of a subject within the guarded perimeter. By following these estimated positions, or waypoints, the subject can be initially intercepted and tracked by the UAV helicopter. Thus, an end goal of this project is the development of a helicopter UAV control system towards waypoint-following.

This type of flight control system can be broken into three coarse components: (1) system modeling and dynamics characterization, (2) feedback-loop flight stabilization, and (3) waypoint-following. The first concern will be the system modeling of the IRIS electric helicopter towards stabilized hovering. Afterwards, the waypoint-following can be implemented as a controlled translation while maintaining a stable hover.

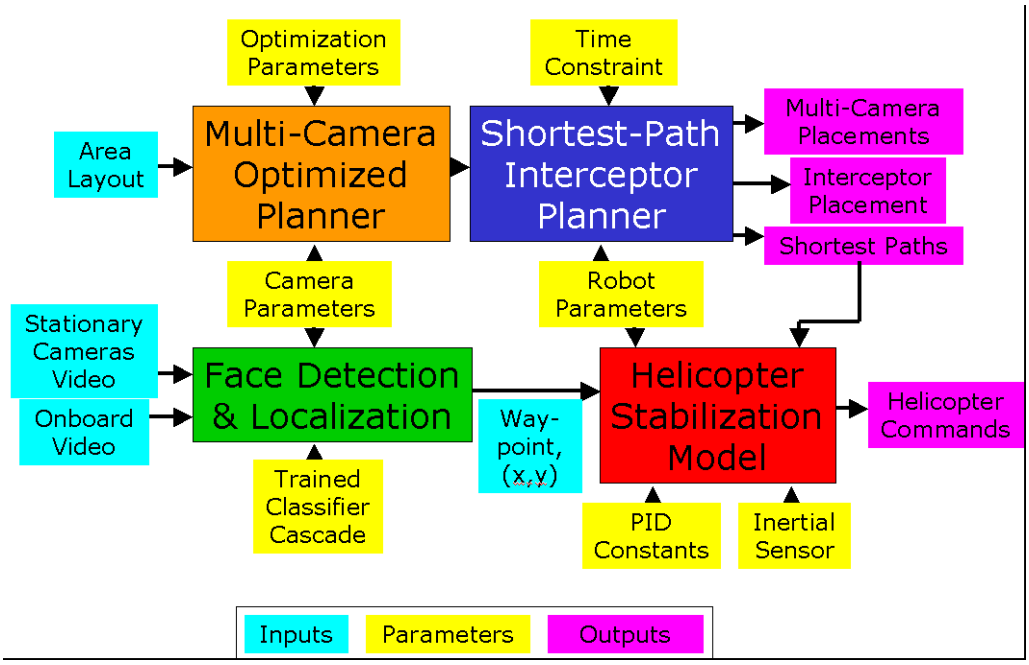


Figure 22. Persistent surveillance system block diagram.

This chapter will first cover sensor placement planning techniques along with a simple, associated method of optimally-placing an interceptor within the area. Next, the helicopter's PID-based flight control system will be described in detail.

3.1 Sensor Planning

Complete surveillance and monitoring of large outdoor areas requires significant manpower, equipment, planning, and the constant vigilance of its guards. Ideally, we would like to replace the individual human responsibilities with automated solutions – minimizing both risk and cost if possible. This section describes a multi-camera system combined with a mobile unmanned air vehicle (UAV) designed to autonomously perform wide-area reconnaissance and intruder tracking in indoor/outdoor environments. This automated solution allows for better overall monitoring with less human involvement.

While a detailed model of real-world environments would produce the most accurate results, three-dimensional modeling is very computationally expensive and data-intensive. Therefore, real-world environments are approximated by their two-dimensional floor plans and layouts. Furthermore, the floor plans themselves can be interpreted as polygons composed of ordered sets of n vertices. These simplifications allow for complex visibility and coverage algorithms to be feasible within reasonable time constraints.

This section will begin by describing two algorithms developed by the author while exploring sensor placement planning. These two algorithms serve as simple methods of generating visibility polygons given a room layout and a viewing point. However, all final experiments are performed with an implemented version of Erdem and Sclaroff's Radial Sweep algorithm.

Next, edge intersection, unique points, and outliers will be examined over two sections. These are problematic issues that can cause errors and crashes in visibility polygon generation.

Camera modeling techniques for both fixed and PTZ cameras comprise the next two sections, with some variations depending on the application. Afterwards, a method for best-mask searching is described along with its variable constraints. Next, shortest-path placement of an interceptor is examined and a solution proposed. Finally, the implementation of a simple OpenCV-based intruder detection and localization system is described.

3.1.1 Vertex Visibility Polygons from Room Layout

Before implementation of Erdem and Sclaroff's Radial Sweep, an attempt was made to write an original vertex visibility polygon algorithm based on sweeping the polygon. This algorithm focuses on finding the visibility polygons for every vertex in the layout polygon instead of an arbitrary point within the polygon. The final goal is to combine the vertex visibility polygon results with the vertex guard results from the Art Gallery solution and achieve a complete coverage map with fewer cameras.

Essentially, the algorithm tries to connect visible line segments with "shadow" projections onto edges behind these segments. Each time a projection is found, the algorithm then "slides" along that intersected edge to the next visible vertex. Ideally, non-visible target vertices are skipped over. Exceptions occur when an edge is intersected

between a target vertex and the main vertex. The algorithm details for each case are explained below.

For each vertex, the algorithm traverses to each other vertex on the perimeter of the polygon in counter-clockwise order, beginning and ending with the main vertex, v(i). Each time the algorithm travels to a target vertex, CV, it checks for intersections with the edge created between the main vertex and the target vertex, and all other edges in the polygon. Successful intersections are inserted into an ascending sorted list, SL, based on radius from the main vertex.

If no intersections are found, then the target vertex must be directly visible from the main vertex. Thus, we can simply push the target vertex onto the visibility polygon list, PV, and set CV to the next vertex on the perimeter (moving in counter-clockwise order). This is the simplest case.

On the other hand, if an intersection is found that is closer to the main vertex than the target vertex, then the target vertex is changed to the start vertex of the closest intersected edge, and a flag is set. This flag will be used to determine the outcome of the next case of the algorithm, when the closest intersection is farther away than the target vertex.

If all intersections are at a greater distance than the target vertex, then that vertex must be directly visible from the main vertex as well. The next visible point should be a "shadow" projection onto an edge behind the target vertex. For most cases, the target vertex and the top of the sorted list (the closest intersection) are both pushed (in that order) onto the visibility polygon list. Also, the target vertex "slides" to the end vertex of the intersected edge, which will be tested for intersections in the next iteration of the algorithm.

However, if the last iteration of the algorithm set a flag, then the two points (target vertex and edge/shadow intersection) are pushed in reverse order (implying the shadow/edge intersection comes before the target vertex in counter-clockwise order). Furthermore, we set CV to the next vertex on the perimeter from the target vertex.

A complete traversal of the polygon perimeter is signaled by arrival back at the main vertex, and should net the visibility polygon. One *caveat* to note: this algorithm was designed and coded on-the-fly, and is sure to have several bugs. Some polygons prove too difficult for the algorithm to converge, and other real-world problems, such as collinear vertices separated by gaps, often confuse the intersection algorithm and cause malfunction. However, adequate results were obtained and used to calculate a complete coverage map for multiple simple polygons, presented in the Results section of this paper.

The pseudo code for the VertexVisibilityPolygons algorithm is presented below. The **insert** function with only two arguments, **insert**(item, list), used in this implementation has two special modifications from normal list insertion functions. First, the function automatically inserts the item into the ascending sorted list at the correct position. Secondly, the function returns the position at which it inserted the item. When three arguments are specified, **insert**(item, position, list), the item is inserted at the specified position as normal.

Algorithm VertexVisibilityPolygons(v, n)	
Input	x- and y-coordinates of the n vertices, v_i, i = 1, 2, …, n, of polygon, P.
Output	Vertex visibility polygons, PV(i), for i = 1, 2, …, n

```
for i = 1 to n
   CV ← next(i)
   flag ← false
   while CV ≠ v(i)
      SL ← NULL
      SVL ← NULL
      PV(i) ← NULL
      visEdge ← edge(v(i), CV)
      for j = 1 to n
         testEdge ← edge(v(j), v(next(j)))
         k ← intersect(visEdge, testEdge)
         if k ≠ NULL
            pos ← insert(k, SL)
            insert(v(j), pos, SVL)
         endif
      endfor
      if SL ≠ NULL
         k ← head(SL)
         s ← head(SVL)
         if distance(k, v(i)) ≥ distance(CV, v(i))
            if not(flag)
               push(CV, PV(i))
               push(k, PV(i))
               CV ← next(s)
            else
               push(k, PV(i))
               push(CV, PV(i))
               CV ← next(CV)
            endif
            flag ← false
         else
            CV ← s
            flag ← true
         endif
      else
         push(CV, PV(i))
         CV ← next(CV)
      endif
   endwhile
endfor
return PV
```

3.1.2 Vertex Visibility Polygons from Visibility Graph

When the vertex visibility graph is already known, finding the vertex visibility polygons becomes an easier task. Since a working visibility graph algorithm was implemented for the shortest-path section of this paper, a second visibility algorithm was created to calculate visibility polygons directly from the visibility graph.

This algorithm works in a similar fashion to the one in the previous section by traversing each of the vertices in counter-clockwise order around the polygon. However, advantage is taken of *a priori* knowledge of the visibility graph. This information is used to narrow down a list of candidate edges with which each target vertex could possibly intersect. Furthermore, only the visible vertices need be considered as target vertex candidates, not all vertices of the polygon. Also, in the trivial case of a completely-visible point, it's immediately known that the visibility polygon is equal to the entire polygon.

In essence, the visibility problem narrows down to a single case where intersections are farther away than the target vertex (since the target vertices are known to be visible). Therefore, we traverse the vertices in order around the perimeter and search for "gaps." A gap is simply a non-visible vertex, or set of vertices, between two visible vertices. Once a gap is found, we intersect the two visible target vertices with all edges

contained within the gap. The closest intersections for each target vertex (if any are found) are then included in the visibility polygon, PV, as shadow projections onto back edges. Either one or both of the surrounding valid target vertices will project onto the back edges.

In summary, this algorithm performs the following operations (presented at a high-level):

1. Initial visible polygon is given by all visible vertices connected in order.
2. Find any gaps in the visible vertices (immediate neighbors not connected in the visible polygon).
3. Fill each gap by intersecting the two visible nodes defining the gap with the edges contained within the gap.
4. Add closest intersected edge for each visible node to visible polygon.

A sample polygon is shown in Figure 23 with colored lines depicting important features. The polygon perimeter is colored in blue. The visibility polygon for the reference point (the leftmost vertex) is colored red. The visibility graph has green lines drawn between the reference point and visible vertices, which are also circled in green. Finally, the gaps between visible vertices are connected with yellow lines. In this example, both of the center visible nodes (connected by green lines) sit on opposite edges of the same gap and both project onto an edge behind. Conversely, the other two gaps only have one visible vertex (those adjacent to the reference vertex) projected onto an edge behind.

Note that this algorithm is still less efficient than the Radial Sweep proposed by Erdem and Sclaroff [5], which will be discussed in the next section. However, using the polar angles and radii of the vertices in the gap to narrow the list of edge-intersection candidates should produce similar efficiency. The full pseudo code for the intersection algorithm, ck = findClosestIntersection(edge, edgeList), used in the visibility algorithm is presented first, followed by the full visibility algorithm, PV = VisibilityPolygonsFromGraph(p, v, n). Note that the intersect function tests each candidate edge against the half-line emanating from point, *p*, with slope equal to that of *refEdge*.

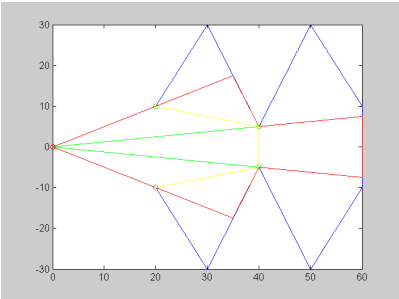


Figure 23. Visibility polygon example.

3.1.3 Edge Intersection

Even the perfectly-defined algorithm stated in text and pseudo code might not transfer easily into a programming language. Several issues with these algorithms in practice came up along the way, and will be discussed below.

The intersect function proved to be difficult to implement since no details were given on its method. The distance of an edge from *x* is directly related to the angle of intersection. In general, the distance decreases as the angle increases between the start vertex of the edge and the closest point. The distance increases as the angle increases between the closest point and the end vertex of the candidate edge. The closest point on the candidate edge occurs when the intersecting edge is perpendicular to it. The non-linearity of the distance as a function of intersecting angle makes edge distance difficult to estimate without calculating the intersection point. For the purposes of the experiment, multiple methods of maintaining the sorted edge list were implemented: distance to the start vertex (given), distance to the end vertex (given), and averaging the distances to the start and end vertices. In general, none proved superior for solving difficult "spiral" polygons. The problem of sorting the edge list accurately remains open in this paper.

The many implementations of a versatile visibility algorithm discussed in this chapter shared many common elements, particularly the heavy-use of edge intersection. Edge intersection is the heart of visibility altogether. This task that's extremely easy for the human eye to accomplish is fraught with difficulty when assigned to a computer. A few practical issues with intersection are mentioned here for relavence.

The foremost problem is rounding-error. Nothing is more frustrating to an implementation author than receiving an intersection back at $(x, y) = (8.275E\text{-}14, 2.435E\text{-}14)$ and having comparisons to $(x, y) = (0, 0)$ fail. Solving these frequent rounding error problems is easiest by setting a minimum tolerance value. Instead of testing for exact equality, the points are tested for distance less than the tolerance.

Occasionally, parallel lines are checked for intersection which, obviously, fails and can crash the implementation. Coincident lines can also frequently cause program crashes. This is another variation of rounding-error, but more difficult to troubleshoot.

When a visible "shadow" falls exactly along an existing edge, the intersection frequently occurs at a vertex. While this occurrence is obvious to the human eye, the algorithm calculates the intersection manually at the vertex coordinates – sometimes creating an extra non-unique visibility vertex with some rounding error. These need to be manually checked against existing vertices at each step, or filtered out during the final "clean-up" pass at the end.

T-intersections can sometimes prove problematic. These occur when a shadow falls in the center of an existing edge. However, the calculated intersection point (with rounding error) can be erroneously interpreted further on as not coincident between the edge's endpoints. If the rounding error puts the new vertex outside the polygon, further sweeps around the polygon can fail and crash the implementation.

This section is only a brief mention of observed edge intersection problems generally caused by rounding error.

3.1.4 Filtering Unique Points and Outliers from the Visibility Polygon

For the implementation of Erdem & Sclaroff's Radial Sweep algorithm [5], the removal of collinear vertices was not included. Instead, all pushed points were saved and post-processed for coherency. Two problems typically occur with the algorithm that can cause potential errors when using the visibility polygon in further calculations: repeated points and outliers. However, the removal of repeated points can fix both problems – if done carefully.

First, since each vertex can listed in the edge list, *Q*, twice, and parts of the algorithm that push a vertex and its shadow push two vertices at once, a single unique edge point can sometimes be listed in the visibility polygon two, three, or even four times – typically in a row. Removing these repeated points from the visibility polygon data structure reduces the unnecessary complexity and redundancy. Additionally, these points might have a small amount of rounding-error if calculated as an intersection; therefore, the points must be compared carefully within a given tolerance.

Algorithm ck = findClosestIntersection(refEdge, edgeList)	
Input	Reference edge, refEdge, and list of potential intersecting edges, edgeList.
Output	Closest intersection, ck, of refEdge and the edgeList.

```
ck ← INFINITY
while not( empty(edgeList) )
   canEdge ← pop(edgeList)
   k ← intersect(refEdge, canEdge)
   if radius(k) < radius(ck)
      ck ← k
   endif
endwhile
return ck
```

Second, nonsensical outliers sometimes crop up in the algorithm in practice. These outliers occur due either to an unfound bug in the implementation or possibly due to the problems of maintaining the sorted list, *SL*, without calculating edge intersections at each step. Currently, a definitive answer to their existence has not been found. However, a post-processing outlier solution can be included in the phase where the visibility polygon data structure is made unique.

The duplicate point- and outlier-removal algorithm begins by checking for duplicate points to the first point of *PV* at the beginning and end of the list and adjusting the list's size so these points are ignored in future calculations. Essentially, index *i* is set to the index of the last duplicate point at the beginning of the list. Index *n* is set to the first duplicate point at the end of the list. The list is then clipped to the portion between indices *i* and *n*.

Next, for each point, p_i, $i \leq n$, in the list, *PV*, we check all points, p_j, $j = i+1, i+2, \ldots, n$, that follow it in the list for duplicates. If a duplicate is found at index *j*, we clip all points between indices *i* and *j* from the list by setting $i = j$. The first part of this algorithm, analyzing the list for duplicates to the first point, p_1, takes linear time. The second part

outer loop takes linear time ($n - d$, where n is the length of PV and d the number of duplicate points of p_1 that were removed). The second part inner loop operates on the portion of the list from index i+1 to n-d, taking logarithmic time. The total complexity is $O(n + n \log n)$, or $O(n \log n)$. Since the original Erdem and Sclaroff Radial Sweep algorithm also takes $O(n \log n)$ time [5], there is no increase in overall complexity.

Algorithm PV = VisibilityPolygonsFromGraph(p, v, n)	
Input	Point, p, and the x- and y-coordinates of the n vertices, v_i, i = 1, 2, …, n, of polygon, P.
Output	Vertex visibility polygon, PV, for point, p.

```
VIS ← VertexVisibility(p, v, n)
PV ← NULL
GEL ← NULL
SV ← head(v)
GSV ← NULL
while not( VIS(SV) )
   SV ← next(SV)
end
CV ← SV
while CV ≠ SV
   if VIS(CV)
      if GEL ≠ NULL
         e ← edge(p, GSV)
         k ←findClosestIntersection(e, GEL)
         if k ≠ NULL
            push(k, PV)
         end
         e ← edge(p, CV)
         k ←findClosestIntersection(e, GEL)
         if k ≠ NULL
            push(k, PV)
         end
         GEL ← NULL
      endif
      push(CV, PV)
   else
      if GEL = NULL
         GSV ← CV
      endif
      e ← edge(CV, next(CV))
      push(e, GEL)
   endif
   CV ← next(CV)
endwhile
return PV
```

3.1.5 Fixed Camera Modeling

For modeling the two primary camera types, an approach based on "probability of detection" was used. Here, a positive detection of an object at point x indicates that the camera is in the proper orientation and foveation to view point x and meet minimum spatial resolution requirements. For fixed cameras, the viewing frustum is fixed at a specific orientation and field of view (or spatial resolution). However, PTZ cameras can change orientation and field of view while online, leading to the concept of "probability of detection."

As stated in the definition of a detection, the probability of detection is simply the probability of the camera being in a valid orientation and lens foveation to meet spatial resolution requirements at each point x. This probability falls off with radial distance from the camera and also angular distance from the original orientation.

Fixed camera modeling is very simple for the most part. For the 2-D layout, the camera mask is a binary combination of three lesser masks: (1) visibility mask, (2) field of view mask, and (3) spatial resolution mask.

The visibility polygon is calculated by Radial Sweep and discretized using an inpolygon() function to form the visibility mask. For calculating field of view and spatial resolution masks, the polar coordinates of the room must be calculated relative to the test point. Then, using the test orientation for reference, all points within half of the maximum field of view on each side of the reference orientation are within the field of view mask.

The spatial resolution uses the polar radius relative to the test point. Using the static field of view for reference, the maximum radius visible to the camera with spatial resolution greater than or equal to the minimum allowable spatial resolution can be calculated. All points within the calculated radius are part of the spatial resolution mask. See Figure 24 [Erdem & Sclaroff, 2005] for an overview of visibility masks.

Algorithm unique(PV, n)	
Input	Visibility polygon list, PV, with n elements
Output	Unique visibility polygon list, VP, with outliers removed

```
VP ← NULL
i ← 1
for j = 2 to n/2
   if PV(j) = PV(1)
      i ← j
   endif
endfor
for j = n/2 to n
   if PV(j) = PV(1)
      n ← j-1
      break
   endif
endfor
while i ≤ n
   push(PV(i), VP)
   for j = i+1 to n
      if PV(i) = PV(j)
         i ←j
      endif
   endfor
endwhile
return VP
```

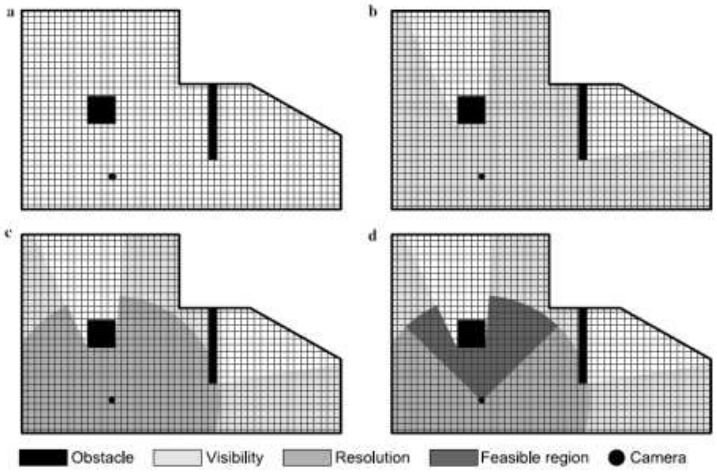


Figure 24. Calculating Visibility for a 2-D Room with Spatial Resolution Consideraiton: (a) typical area with camera point; (b) visibility mask; (c) maximum spatial recognition radius; and (d) field of view.

Two other considerations can also come into play, depending on the system goals. The first is camera tilt. If the cameras are always assumed to be facing straight out at face level, the 3-D room can be interpreted simply as a 2-D plane at the mounted height. However, many real-world situations call for the cameras to be placed higher. In these cases, the camera's 3-D viewing frustum can be projected onto the ground plane (at height, *h*) and the radius and spatial resolution masks can be calculated using the new projected view.

Figure 25 depicts simple diagrams of camera tilt and projection. The total coverage for each point in the room can be calculated as follows:

1. Simple 3-D projection of image plane onto the floor plane.
2. Visibility based on vertical and horizontal fields of view, tilt angle, and height from floor.
3. Probability of coverage based on spatial resolution (linear).

Secondly, the minimum spatial resolution might not be a hard limit as depicted above. In this case, the camera mask is not binary – instead, it can be interpreted as a "probability of coverage," where the values range from 0 to 1 depending on the probability that an intruder will be detected using the given location and camera parameters. This concept will be explored further in the PTZ camera modeling.

For fixed cameras, "probability of coverage" could simply mean the extension of the coverage past the previous hard limit at the minimum spatial resolution radius. The probability of detection remains one (or true) up to the hard limit, at which point it drops off according to some function of the radius as it approaches infinity. Good candidates for this function are linearly-decreasing or exponential.

The resulting equations are presented as Equation 1. Here, the mounting height and tilt angles are constant and pre-determined. Camera parameters such as maximum field of view (β_{FOV}) also play a part in determining the maximum viewable range to meet spatial recognition requirements.

Equation 1. Calculating Coverage for a Tilted Fixed Camera

$$H = const, \varphi_{TILT} = const$$

$$\phi = \theta - \theta_{BIS}, -\pi \le \phi \le \pi$$

$$r_n = H \tan\left(\varphi_{TILT} + \frac{\beta_{TILT}}{2}\right), r_f = H \tan\left(\varphi_{TILT} - \frac{\beta_{TILT}}{2}\right)$$

$$x_n' = \frac{r_n}{\cos\phi}, x_f' = \frac{r_f}{\cos\phi}$$

$$r' = \sqrt{r^2 + H^2}$$

$$v \in \left\{\left(x_n' \le r \le x_f'\right) \wedge \left(|\phi| \le \frac{\beta_{FOV}}{2}\right) \wedge \left(r' \le \frac{p}{SR \cdot \beta_{FOV}}\right)\right\}$$

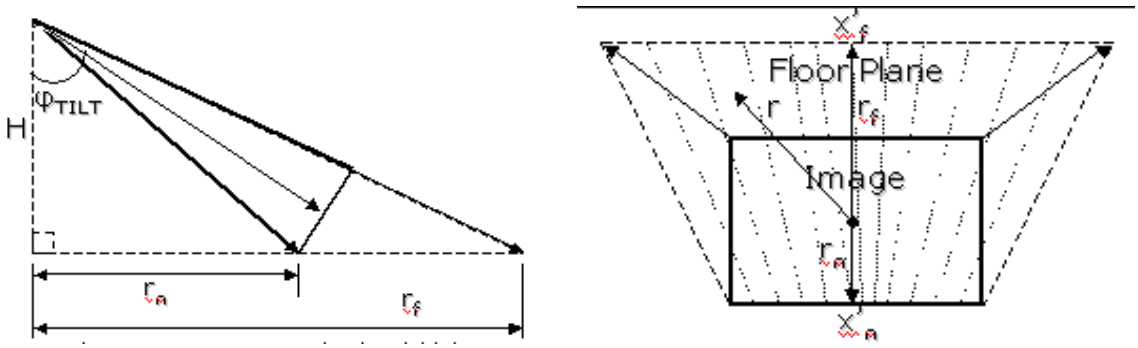


Figure 25. Accounting for tilt when modeling fixed cameras: (a) side view, and (b) overhead view.

3.1.6 Pan-Tilt-Zoom (PTZ) Camera Modeling

For pan-tilt-zoom cameras, probability of coverage comes in to play in the majority of placements. Two methods of PTZ camera modeling are presented below for two different scenarios. In the first, the PTZ camera is assumed to pan back and forth at a constant rate over the entire viewing range. Since cameras are assumed to be mounted on walls only, the viewing range is always the edge angle at the test point. In the second case, the camera is allowed to pan/tilt to random positions according to a Gaussian probability distribution. The unpredictable behavior of the camera makes it much more difficult for an intruder to slip past.

For the simpler case of constant panning, a few considerations must first be made. The first is the radius of the farthest point within the visibility mask. Knowing this distance, the minimum field of view can be calculated. Furthermore, knowing the total viewing angle of the visibility mask and using the calculated minimum field of view as the viewable portion at any given time, the pan ranges and final probability of coverage can be calculated.

Equation 2. Simple PTZ Modeling

$$\theta_{FOV} = \min\left(\frac{p}{r \cdot SR}\right), \alpha_{FOV} \le \theta_{FOV} \le \beta_{FOV}$$

$$\phi = \theta - \theta_{BIS}, -\pi \le \phi \le \pi$$

$$\theta_{Cover} = \min(\beta_{EDGES}, \beta_{PAN} + \theta_{FOV})$$

$$\theta_{PanCCW} = \phi + \theta_{FOV}, \theta_{PanCCW} \le \frac{\theta_{Cover}}{2}$$

$$\theta_{PanCW} = \phi - \theta_{FOV}, \theta_{PanCW} \ge -\frac{\theta_{Cover}}{2}$$

$$p_{PAN} = \frac{\theta_{PanCCW} - \theta_{PanCW}}{\theta_{Cover}} = \frac{2\theta_{FOV}}{\theta_{Cover}}, \left|\phi \pm \theta_{FOV}\right| < \frac{\theta_{Cover}}{2}$$

The probability that each point will be visible at any given time can be calculated as follows:

1. Find minimum zoom to view each point at minimum spatial resolution requirement.
2. Find pan/tilt ranges around point using calculated zooms.
3. Calculate probability of being in a valid P·Z for each point.

For the second case of a random orientation camera, the process is very similar. However, a Gaussian distribution is applied instead of the linear mapping of the probability distribution within the edge angle. The center position is preferred and is therefore the mean of the distribution. The standard deviation of the distribution is adjusted to encompass the entire edge angle range 99.7% of the time (three standard deviations). A similar process can be applied to both pan and tilt ranges, depending on the application.

The probability that each point will be visible at any given time can be calculated as follows:

1. Find minimum zoom to view each point at minimum spatial resolution requirement.
2. Find pan/tilt ranges around point using calculated zooms.
3. Calculate probability of being in a valid P·T·Z foreach point.

Furthermore, a two-step detection system can instead be assumed. In this case, the first step is motion detection through optic flow. Detected motion causes the PTZ camera to foveate on the location (increasing spatial resolution). Object detection can then be performed on the higher resolution images. For this system, the cameras are allowed to remain at the maximum field of view that would still allow for adequate motion detection. However, the total coverage encompasses all pan-tilt-zoom combinations that meet minimum spatial resolution requirements.

Equation 3. Zoom Terms for Probability-based PTZ Camera Modeling

$$\theta_{FOV} = \frac{p}{r \cdot SR}, \alpha_{FOV} \le \theta_{FOV} \le \beta_{FOV}$$

$$\mu_{FOV} = \beta_{FOV}, \sigma_{FOV} = \sqrt{\frac{\max(\theta_{FOV}) - \min(\theta_{FOV})}{3}}$$

$$p_{FOV} = 1 + erf\ \frac{\theta_{FOV} - \mu_{FOV}}{\sqrt{2}\sigma_{FOV}}$$

Equation 4. Pan Terms for Probability-based PTZ Camera Modeling

$$\phi = \theta - \theta_{BIS}, -\pi \le \phi \le \pi$$

$$\theta_{PanMax} = \min(\beta_{EDGES}, \beta_{PAN})$$

$$\theta_{PanCCW} = \phi + \theta_{FOV}, \theta_{PanCCW} \le \frac{\theta_{PanMax}}{2}$$

$$\theta_{PanCW} = \phi - \theta_{FOV}, \theta_{PanCW} \ge -\frac{\theta_{PanMax}}{2}$$

$$\mu_{PAN} = \theta_{BIS}, \sigma_{PAN} = \sqrt{\frac{\theta_{PanMax} - \min(\theta_{FOV})}{3}}$$

$$p_{PAN} = \frac{1}{2}\left[erf\left(\frac{\theta_{PanCCW} - \mu_{PAN}}{\sqrt{2}\sigma_{PAN}} \right) - erf\left(\frac{\theta_{PanCW} - \mu_{PAN}}{\sqrt{2}\sigma_{PAN}} \right) \right]$$

Equation 5. Tilt Terms for Probability-based PTZ Camera Modeling

$$\varphi_{FOV} = 2\tan\left(\frac{3}{4}\arctan\left(\frac{\theta_{FOV}}{2} \right) \right), H = const$$

$$\varphi = \arctan\left(\frac{H}{r} \right), 0 \le \varphi \le \frac{\pi}{2}$$

$$\varphi_{TiltMax} = \min\left(\frac{\pi}{2}, \beta_{TILT} \right)$$

$$\varphi_{TiltDown} = \varphi + \varphi_{FOV}, \varphi_{TiltDown} \le \frac{\varphi_{TiltMax}}{2}$$

$$\varphi_{TiltUp} = \varphi - \varphi_{FOV}, \varphi_{TiltUp} \ge -\frac{\varphi_{TiltMax}}{2}$$

$$\mu_{TILT} = const, \sigma_{PAN} = \sqrt{\frac{\varphi_{TiltMax} - \min(\varphi_{FOV})}{3}}$$

$$p_{TILT} = \frac{1}{2}\left[erf\left(\frac{\varphi_{TiltDown} - \mu_{TILT}}{\sqrt{2}\sigma_{PAN}} \right) - erf\left(\frac{\varphi_{TiltUp} - \mu_{TILT}}{\sqrt{2}\sigma_{PAN}} \right) \right]$$

3.1.7 Best Mask Combination Searching

First the visibility masks are calculated, followed by several camera masks for each test point (corresponding to different camera types and orientations). Next, the entire mask set can be searched for the best combination that yields optimized characteristics according to the input constraints.

The (typically) foremost important constraint is coverage. Usually only a minimum percent coverage amount is specified, such as 90%. A point is “covered” if it lies within at least one camera mask within the solution set. The sum of the covered

points divided by the sum of all in-polygon points yields the coverage. Once the coverage requirements are met, optimization attempts to minimize other characteristics.

The second constraint typically to be minimized is cost. Cost can be interpreted as a monetary amount, a flat-rate per camera (minimizing the number of cameras), bandwidth usage per camera, etc. The typical optimization goal of sensor planning is cost minimization and coverage maximization.

Probability of detection is another important search constraint. Each point in each camera mask is assigned a probability of being covered at any given time according to camera modeling. In the solution mask set, the overall probability of coverage for each point can be analyzed for feasibility. The probability can be thresholded to determine if a point is covered. The probability can be combined at each point for multiple cameras, yielding an overall probability that at least one camera can "see" the point at any given time. Or, the probabilities can simply be added over the entire solution mask to determine overall coverage. However, including probability of coverage requires a slightly different algorithm with much higher storage requirements and computational complexity (floating point versus binary operations).

Worst-case foveation time (WCFT) is simply a measure of a given camera mask's ability to foveate on any given point at any given time. Actually, two worst-case situations can be examined. In the first, the camera's starting orientation is always at the default (typically, the edge bisector). The worst case foveation time is simply the time to pan to the farthest edge. In the second case, the camera can be in any starting orientation. The worst-case time here is generally the time to pan from one edge to the other. Basically, it allows a hard-limit on how fast an intruder can move through the scene and still be detected. Limiting the WCFT to zero essentially disallows panning completely.

- Each node in the search tree is dynamically generated and comprised of
  - Set of m masks, where m equals the depth of the node
  - Binary union (OR) of all masks in the set
  - Total cost of all masks, C
  - Worst-case foveation time, max(T)
- Minimum Coverage, S
  - S=0: Any minimized coverage is valid.
  - S>0: Limit minimum coverage for a valid solution
- Maximum Cost, C
  - C<0: Ignore cost during search
  - C>0: Limit cost for a valid solution
  - $C \rightarrow \infty$: Any minimized cost is valid
- Maximum Foveation Time, T
  - T<0: Ignore foveation time during search
  - T=0: Foveation time must be zero (fixed cameras only)
  - T>0: Limit foveation time to T seconds.
  - $T \rightarrow \infty$: Any minimized foveation time is valid.
- Probability of Coverage, P
  - P=0: Any probability of coverage is valid.

- P>0: Coverage probability must be greater than threshold, P.
- P=1: Must have 100% probability (fixed cameras only).

Branch-and-bound searching is a textbook method for making an optimized set of choices and is particularly useful for combinatorial problems. Essentially, the solution space is searched exhaustively while unpromising avenues are pruned. The implementation used here is fairly generic. All possible combinations of masks are explored in a breadth-first manner. Avenues that don't meet constraints (cost is higher or coverage is lower than the best set found so far) are pruned from the search tree.

Algorithm pseudo code is below. For sensor planning, the inputs would include a set of visibility masks and associated costs and the desired coverage. The output would include the minimized cost to meet coverage requirements and the corresponding camera mask subset.

3.1.8 Shortest-Path Interceptor Placement

A sub-problem of sensor planning for the purposes of this automatic surveillance system utilizing an interceptor robot is finding the best location for the robot to be located. Thus, the time to intercept an intruder at any entry point is minimized. Here, the maximum allowable time-to-intercept and speed of the robot are given. Thus, the maximum radius of interception can be calculated.

Algorithm bfs(v, depth)	
Input	Vertices, v, constraints, costs, and max depth
Output	Optimized vertices combination to meet constraints

```
Initial Call
    maxDepth := length(v)
        for depth := 1, 2, …, maxDepth
            bfs(v, depth)
Simplified Algorithm
    bfs(v, depth)
        if process(v) meets min. reqs. OR depth = 0
            maxDepth := depth
            return
        mark v as visited
        for all vertices i adjacent to v not visited
            bfs(i, depth-1)
```

Shortest-path calculation is an extension of visibility combined with graph-searching. Once the vertex visibility graph is obtained for a polygon, edges are drawn between all visible nodes with weight equal to Euclidean distance. Finding the shortest path between a single vertex and all other vertices on the perimeter is then simply a matter of applying Dijkstra's greedy algorithm [Dijkstra, 1959].

In an automatic surveillance system where a robot is available for intruder interception, one important task is determining the best location to place the robot within the monitored area. Assuming the robot intercepts intruders by traveling to or gaining line

of sight on the point of intrusion, the goal is to minimize the distance to all points on the perimeter. In the case of a star convex polygon (having a point or points in the interior that are visible from any point on the perimeter), we can place the robot at a star convex point and achieve Euclidean distance to all points on the boundary. However, few real-world areas will be star convex. Therefore, this experiment calculated the shortest paths between all vertices using the weighted visibility graph (obtained using the algorithm in section 4.1) and Dijkstra's algorithm. Furthermore, the centroid of the polygon was tested for viability as the best placement for the robot in cases where the centroid lies within the polygon.

The results of the shortest path calculation will be used throughout this section in an attempt to find the best placement for one or more robots within the monitored area. The centroid is the center of mass for a polygon. In essence, the centroid is the X-Y balance point for an object in the real-world. Informally, it can be considered the "average" of all points within the object. The centroid of a polygon with N vertices can be calculated using simple equations. The equations are implemented in the centroid algorithm.

Equation 6. Centroid of a Polygon

$$A = \frac{1}{2}\sum_{i=0}^{N-1}(x_i y_{i+1} - x_{i+1} y_i)$$

$$c_x = \frac{1}{6A}\sum_{i=0}^{N-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$c_y = \frac{1}{6A}\sum_{i=0}^{N-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

Algorithm p = centroid(PV)	
Input	Polygon, PV
Output	Centroid, p, of the polygon, PV

```
n = size(PV)
A = 0
cx = 0
cy = 0
for i = 1 to n
   m = (PV(i)→x) * (next(PV(i))→y) – (next(PV(i))→x) * (PV(i)→y)
   A = A + m
   cx = cx + ((PV(i)→x) + (next(PV(i))→x)) * m
   cy = cy + ((PV(i)→y) + (next(PV(i))→y)) * m
endfor
A = A / 2
cx = cx / (6 * A)
cy = cy / (6 * A)
p = point(cx, cy)
return p
```

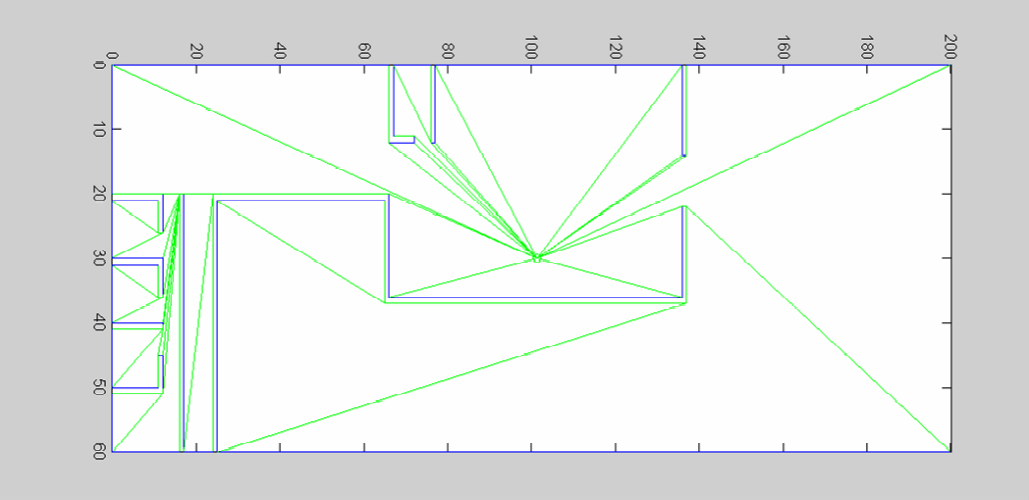


Figure 26. Shortest paths from the centroid of a polygon (IRIS West).

The centroid formulas were used in experimentation to find the centroid of the entire polygon, and also the visible and vertex-visibility polygons. With the centroid's natural “average” distance to the perimeter and visibility greater than or equal to any other point on the perimeter (and most points within), it provides an interesting candidate point for final placement. Pseudo code for centroid calculation is also presented below.

The robot is not allowed to move directly to the intruder upon detection. Instead, the shortest-path route determined by the interior of the polygon must be used. Otherwise, the centroid of the polygon would provide the natural best location for a robot able to move through walls.

Calculation of the shortest-path can be performed using Dijkstra’s algorithm [Dijkstra, 1959] on the weighted graph. The weighted graph is simply the visibility graph of all test points and the Euclidean distance between them.

Because a discrete H x W layout of the room was created for sensor planning, digital image processing can be applied to the layout with interesting effects. Specifically, morphological operations can be applied to assist in finding the best placement for an interceptor robot. Since many morphological operations require binary images anyway, the discrete layout is well-suited to these methods.

Finding the skeleton of the layout image is essentially finding the medial axis. The boundary is shrunk inwards at a constant rate until collapsed into lines. For images, this process is the iterative application of the skeletonization formula. Or, the image can be collapsed down to a single point, known as “shrinking.” Both provide interesting test points for the shortest-path interceptor placement. In the continuous domain, finding the skeleton is less computationally expensive overall. The Voronoi diagram of the polygon produces the straight skeleton. Once the straight skeleton is found, samples can be taken and used for test points. Figure 27 depicts a simple interception mask around an obstacle.

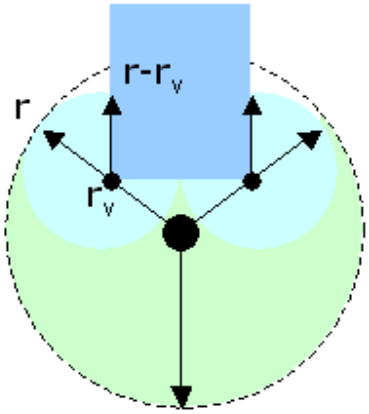


Figure 27. Calculating the intercept mask around an obstacle.

The process for constructing shortest-path masks can be viewed graphically. This process is much like finding camera masks with 360º field of view where the interception radius is equivalent to the spatial recognition radius. However, for interception, the vehicle can follow corners around the interior boundary.

1. Calculate the Voronoi diagram, or straight skeleton, of the (bounded) polygon.
2. Sample all intersections and a few points along the line segments.
3. Construct the shortest-path masks for these test points.
4. Perform the branch and bound searching to find the best placement(s).

In Figure 27, the vehicle has an interception radius, *r*, to the edge of the green outer circle. However, the obstacle must be avoided. By simply pre-calculating the visibility masks (blue circles) from all vertices and using Dijkstra's shortest-path algorithm at each test point, the total interception mask simply becomes the union of all visibility masks within radius, *r*, of the test point, including the test point's own visibility mask. Therefore, the shortest-path masks algorithm can be summarized in 6 steps.

1. Pre-calculate the visibility polygons and masks for all interior vertices of the area polygon.
2. Pre-calculate the shortest-paths between all vertices using Dijkstra's algorithm.
3. For each test point, calculate the visibility polygon and mask.
4. Limit the test point's visibility mask to radius, *r*, and store in the interception mask set.
5. Calculate the shortest-paths to all area vertices using Dijkstra's algorithm.
6. For each vertex, *v*, within radius, *r*, limit the vertex's visibility mask to radius, $r - r_v$ (where r_v is the distance traveled from the test point to vertex, *v)*, and union the resulting mask with the interception mask set.

3.1.9 Object/Face Recognition and Tracking

The face localization experiments deal with detecting and estimating the location of a person within a known area. These experiments were performed in conjunction with the larger project of camera placement planning with automatic intruder detection and a UAV helicopter interceptor/tracker. In essence, the system consists of three parts: (1) a face detection algorithm, (2) location estimation using uncalibrated camera field-of-view and average face-width, and (3) final Kalman location estimation.

Testing the sensor planning and interceptor placement algorithms in the real world requires one more component: intruder detection and localization. For the purposes of this experiment, the face of a human intruder is the target object. In order to maximize the probability of face recognition, the cameras are placed at eye level (as assumed in the sensor planning). While overhead cameras would provide much better localization to the 2-D ground plane, they also provide very poor facial recognition results. Figure 28 depicts the overall system. The system algorithm is listed below.

1. Capture an image frame from live video web camera
2. Apply face detector to image and return bounding rectangles of faces within the image
3. Use simple projective transform to estimate distance (r) and angle (θ) with respect to the camera. Convert polar coordinates to (x, y) pair.
4. Perform Kalman update to better estimate position.
5. Perform Kalman predict to estimate next position.
6. Loop to 1st step.

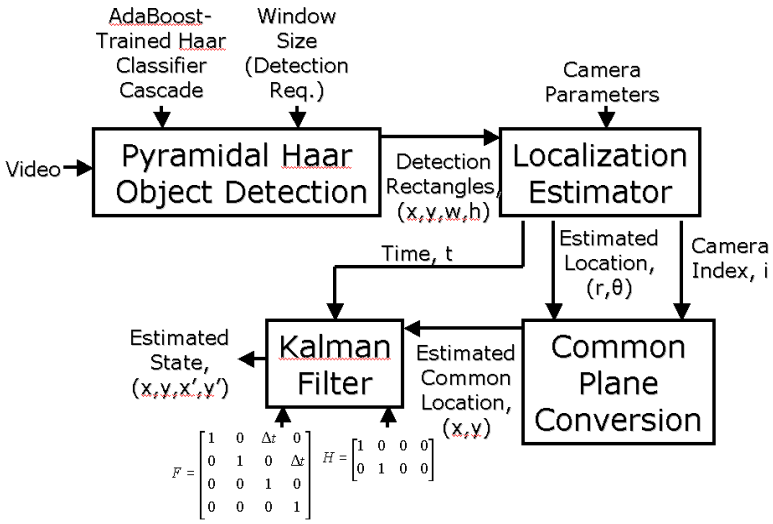


Figure 28. Object/face detection and localization block diagram.

Face recognition was performed using Intel OpenCV. The object descriptor implemented was initially proposed by Paul Viola [Viola01] and later refined by Rainer Lienhart [Lienhart02]. The classifier is a "cascade of boosted classifiers working with haar-like features."

For testing of frontal face detection, a trained classifier cascade included with the OpenCV was used, "haarcascade_frontalface_alt2.xml." This particular detector is a tree-based 20x20 gentle AdaBoost frontal face detector, also created by Rainer Lienhart. Thus, the minimum required (ideal) spatial resolution is 20 pixels horizontally and vertically for sensor planning.

Localization from a camera on the perimeter mounted at approximate eye-level to a subject can be estimated from the camera's current field of view and the width of the detected face (in pixels) within the captured images. Instead of relying on lens calibration to provide a good estimate of the target's location, a very coarse estimate will be provided and improved through Kalman filtering.

Because the maximum fields of view for the camera, β_{FOVh} (horizontal) and β_{FOVh} (vertical), are already known from sensor planning, those values will be used in determining the location of the detected face within the X-Y plane. Using simple geometry and an estimate of average face width, F_w, the polar coordinates can be determined with respect to the camera axis. The detection rectangle within an image, *I*, is defined as *R*. Subscripts *w* and *h* refer to width and height, respectively. The subscript *c* refers to the object's center. The equations for determining the estimated polar coordinates of the detected face are listed below.

Equation 7. Face localization from detection rectangle.

$$R_c = R_x + \frac{R_w}{2}$$

$$\theta_h = \frac{R_w \beta_{FOVh}}{2 I_w} + \arctan\left(\frac{4}{3}\tan\left(\frac{R_h \beta_{FOVv}}{2 I_h}\right)\right)$$

$$r = \frac{F_w}{2\sin\left(\frac{\theta_h}{2}\right)}$$

$$\theta = \beta_{FOVh}\left(\frac{R_c}{I_w} - \frac{1}{2}\right)$$

The localization estimate can be greatly improved through the use of Kalman filtering – provided that several frames of the detected face are available. For position estimation, we only need a four-state Kalman filter: x-position, y-position, x-velocity, and y-velocity. No controls are included in the system. Thus, the equations and Kalman matrices can be summarized below.

Equation 8. Kalman Predict

$\mathbf{x} = \mathbf{Fx} + \mathbf{Bu}$ (Predicted State)
$\mathbf{P} = \mathbf{FPF}^{T} + \mathbf{Q}$ (Predicted Estimate Covariance)

Equation 9. Kalman Update

$\mathbf{y} = \mathbf{z} - \mathbf{Hx}$ (Innovation or Measurement Residual)
$\mathbf{S} = \mathbf{HPH}^{T} + \mathbf{R}$ (Innovation or Residual Covariance)
$\mathbf{K} = \mathbf{PH}^{T}\mathbf{S}^{-1}$ (Optimal Kalman Gain)
$\mathbf{x} = \mathbf{x} + \mathbf{Ky}$ (Updated State Estimate)
$\mathbf{P} = (I - \mathbf{KH})\mathbf{P}$ (Updated Estimate Covariance)

Equation 10. Kalman Matrices

$$x = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix},$$

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$Q = 0.1I_{4x4},$$

$$u = 0,$$

$$B = 0,$$

$$z = \begin{bmatrix} x_m \\ y_m \end{bmatrix},$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

$$R = I_{2x2},$$

$$P_0 = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix},$$

$$x_0 = \begin{bmatrix} x_{m0} \\ y_{m0} \end{bmatrix}$$

3.2 Proportional-Integral-Derivative (PID) Control System

PID feedback loops provide the primary automatic control of the helicopter. The primary advantage of PID control is its low complexity and robust behavior for real-time operation. The core control system is comprised of two primary loops: an inner-loop PID attitude controller, and an outer-loop PID guidance controller (waypoint navigation). Additionally, various methods of PID tuning and tweaking towards the UAV helicopter application will be examined in this section. Next, the controls/servo mixing required to translate PID outputs into raw servo commands will follow. The section will conclude with descriptions of the two final control systems and their corresponding hardware configurations.

3.2.1 Hardware

The hardware components of the UAV helicopter flight control system include the helicopter itself, standard R/C components, onboard orientation sensors, and servo controllers.

The IRIS helicopter is designed and manufactured by Miniature Aircraft, USA. The model is a .90 size XCell ION-X^2 [Miniature Aircraft, 2007] electric helicopter. However, the stock rotor head assembly has been replaced with the manufacturer's similar Tempest 3D model [Miniature Aircraft, 2007]. The main rotor blades are SAB 810-mm S-cambered (art. 0231) [SAB Composites, 2007].

The electric motor speed controller is a schulze future-40.160WK [schulze electronik, 2007]. The motor itself is a Köhler actro Compact self-cooling brushless DC [actro, 2007]. Swash-plate servos are all JR DS8311 digital servos with a JR R649 PCM 9-channel receiver (part of the complete JR XP9303 Heli package [JR Radios, 2007]). However, the high-speed tail servo, actuated through the gyro, is a Futaba S9254. The gyro itself is a Futaba GY401 Angular Vector Control System (AVCS) rate gyro (PI control) [Futaba, 2007]. Finally, the transmitter used is a JR XP9303 9-channel PPM/PCM programmable model. Figure 29 depicts the helicopter system.

(a) (b)

Figure 29. (a) Stock Xcell ION-X Electric Helicopter, and (b) JR XP9303 Transmitter.

The FMA Co-Pilot CPD4 [FMA Direct Co-Pilot, 2007] is a flight stabilization system based on infrared signatures. Essentially, before lift-off, the infrared signature of the ground and the sky are calibrated into the CPD4 unit. Once in air, the CPD4 constantly monitors its position between the earth and the sky. If the transmitter control sticks are centered at any time, the CPD4 automatically levels the helicopter according to the infrared signatures (making sure the sky is directly up and the earth is directly down).

Control of the helicopter is made by hooking the unit in series from the helicopter's receiver to the two swash-plate servos. Thus, the unit behaves similar to a gyroscope – except in the vertical pitch axes rather than for yaw.

Unfortunately, the original purchased unit was found to be incompatible with our helicopter. The IRIS helicopter uses a more complicated 120º Cyclic/Collective Pitch Mixing (CCPM) [Heli Hobby, 2007] swashplate-configuration. Instead of using one servo to pitch the swashplate left or right, and one servo to pitch it forward and backwards, the CCPM configuration uses three servos positioned 120º apart from one another. One servo is situated in the front of the helicopter, with the others to the sides and behind the swashplate. Thus, moving the swashplate in a given direction requires mixing the right amount of movement of each servo.

Since the CPD4 only supported traditional two-servo configurations, it was not applicable to the project. The unit was returned to the manufacturer for a refund of the purchase price ($99.95), to be used towards the purchase of a newer, compatible model. Figure 30 depicts the Co-Pilot systems [FMA Direct Co-Pilot, 2007].

Instead of the standard receiver included in the JR package, the FMA Co-Pilot FS8 is used. Along with the included vertical and horizontal infrared sensors (six in all), the unit is capable of estimating orientation in real-time based on the thermal signatures of the earth and sky. Using this information, the helicopter's attitude can be automatically leveled with respect to the ground plane. However, the unit only takes control with the cyclic pitch (right stick on the transmitter) is released by the pilot. This behavior allows for full control of the helicopter with assistance only when needed.

(a) (b)

Figure 30. FMA Co-Pilot System: (a) CPD4 and (b) FS8 package [FMA Direct Co-Pilot, 2007].

Presumably, the Co-Pilot simply uses a PI/D control loop with the infrared sensor readings as feedbac. However, this only works properly outdoors at a safe distance from buildings, trees, etc. Otherwise, the infrared signatures of surrounding objects interfere with the attitude estimation. Moreover, the unit must be carefully calibrated to the maximum servo throws, directions, and sensor mounting orientations. The sensors must also be recalibrated before each flight and in the event of weather changes.

The Co-Pilot has two additional features that prove useful in research. First, the real-time control, servos, and orientation data can be read from the transmit pins attached to the receiver. This data is transmitted in digital serial/RS-232 format, compatible with a standard PC. Thus, the onboard flight computer can read and log this data in real time using the FMA Viewer software. Secondly, proprietary digital signal recognition (DSR) technology guarantees less interference from neighboring frequencies, and failsafe servo positions can be programmed into the Co-Pilot for instances where transmitter communication is lost. Overall, the Co-Pilot provides a researcher with increased ease of flight along with an easy means of logging flight data for experimental use.

The Xsens MT9-B [Xsens MT9, 2007] inertial measurement unit (IMU) is a complete small form factor system with 3-axis accelerometers, gyroscopes, magnetometers, and a single temperature sensor. Using proprietary sensor fusion techniques built into the unit, the unit transmits real-time filtered orientation along with calibrated and raw sensor readings via serial/RS-232. The system specifications can be seen in Table 4.

Table 4. Xsens MT9-B Specifications

		rate of turn	**acceleration**	**magnetic field**	**temperature**
Unit		*[deg/s]*	*[m/s/s]*	*[mGauss]*	*[°C]*
Dimensions		3	3	3	-
Full Scale	*(units)*	+/- 900	+/- 20	+/- 750	-55…+125
Linearity	*(% of FS)*	0.1	0.2	1	<1
Bias stability	*Compensated*				
	(units 1σ)	5	0.02	0.5	-
	Uncompensated				
	(units per °C)	1	0.02	-	-
Scale factor	*Compensated*				
stability	*(% 1σ)*	-	0.05	0.5	-
	Uncompensated				
	(% per °C)	0.15	0.03	0.5	-
Noise	*(units RMS)*	0.7	0.01	4.5	0.0625
Alignment error	*(deg)*	0.1	0.1	0.1	-
Bandwidth	*(Hz)*	50	30	10	-

The attitude-heading reference system (AHRS) is primarily self-contained within the Xsens MT9-B IMU. However, later in this section, the operation of a typical AHRS based on accelerometers, gyroscopes, and magnetometers will be examined. The behavior of the AHRS has a large impact on the inner-loop attitude control described later in the next section on PID control. Essentially, the AHRS estimates the orientation of the helicopter.

The Xsens MT9-B uses a proprietary, undocumented sensor fusion algorithm for estimating the orientation of the IMU [Xsens Technical Documentation, 2004]. This algorithm uses measurements of gravity and earth's magnetic field to compensate for the drift caused by integrating the rate-of-turn measurements. Unfortunately, situations exist that can cause this algorithm to perform poorly or fail altogether.

One of these situations, learned through experimentation, is high vibration. In the experiments section of this paper, one section will focus on obtaining the best orientation estimate from the MT9-B by adjusting both mounting methods and sensor fusion algorithm parameters.

Two parameters can be specified for the sensor fusion algorithm: filter gain (or accelerometer/magnetometer "crossover" frequency) and magnetometer weighting. Typically, both are set to values of 1.0 (frequency in Hz and unit-less gain, respectively). The filter gain basically determines the frequency "crossover" between relying on accelerometer/magnetometer (gravity/earth's magnetic field) data and gyroscope (rate-of-turn) data. Low-frequency data, below 1.0 Hz, is considered stable enough for compensation of the rate-of-turn integration. High-frequency data, above 1.0 Hz, is preferred for the gyroscopes – since anything below this threshold is more than likely bias error. Much about the Xsens MT9-B will be learned later during the control system's flight-testing experiments phase.

In order to mount equipment and increase the flight safety of the helicopter, oversized landing skids were attached to the standard gear. The gear is designed in two pieces – an upper component mounting cage and lower training skids. Both share the same ¼" aluminum bar design. Two identical legs are simply crossed to create the complete training gear. Figure 31 depicts the training gear design.

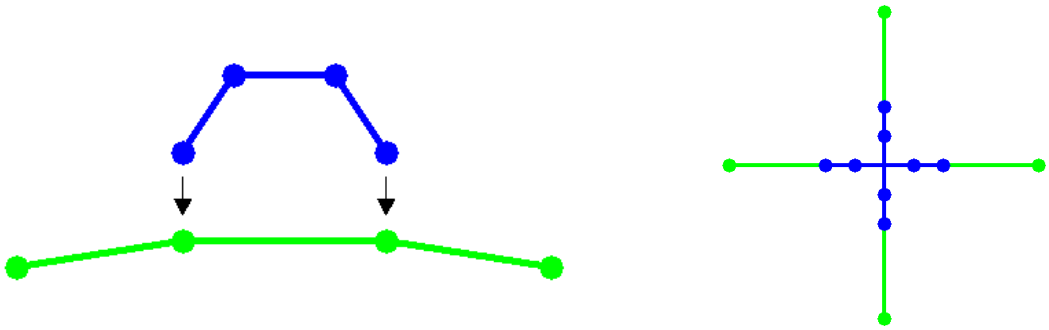


Figure 31. (a) Side view of single training gear leg with upper cage leg mounted above; and (b) Overhead view of complete training gear assembled.

Two distinctly different servo control schemes were used in this project. Both have distinct advantages and disadvantages. The Endurance R/C PCTx [Endurance R/C, 2007] excels in controlled experimentation where the helicopter is tethered and safety is the primary concern; however, it is not a practical final setup for free-flight. The Pololu 16-servo Controller [Pololu, 2007] directly controls the servos at a high-resolution. However, this requires replacing the standard R/C components and using a PC, instead. While this is an eventual goal, replacing the standard R/C components prevents manual piloting during testing.

The Pololu USB 16-servo controller has several significant advantages over other methods, including its ease of use and interfacing, different programming modes, dual USB and UART interface, and ability to control up to 16 servos. An image of the servo controller, with callouts, is shown below. The controller can be interfaced from Windows XP using the mini-B USB port, or from any UART interface using the pins along the left edge. Additionally, the servo controller requires an external power source dedicated to the servo power lines. With unconnected servo power lines, the controller only sends a control signal on the signal line to each servo.

The Endurance R/C PCTx was purchased directly from the company for around $50 USD. The unit connects to a standard PC via USB and communicates with an R/C transmitter through its trainer port. This gives cheap, reliable PC control of any standard R/C vehicle without tearing down its manual control system. The PCTx supports up to nine channels. Additionally, the PCTx has a 50Hz refresh rate, independent servo control, adjustable pulse width, and C++, C#, and VB.NET software API's available.

The primary advantage of using the PCTx is the ability to move most components off-board during flight-testing. Furthermore, the system is easier to transition between manual and automatic control. Finally, the pilot can act as a near-instant failsafe in the result of flight computer error. These advantages add up to a system that is far superior to the Pololu version for testing purposes, albeit not as practical for a final system design. Figure 32 depicts the two servo controllers.

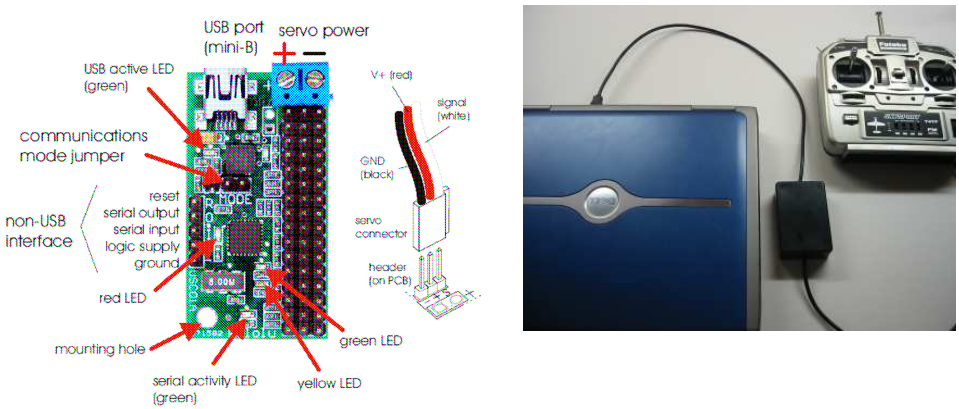


Figure 32. Servo controllers: (a) Pololu USB 16-servo controller and (b) Endurance R/C PCTx connected.

3.2.2 PID Control Theory

PID controllers are used in many industrial applications that require feedback control of a non-linear process. A beginning overview of PID control can be found in [Sellers, 2002]. The traditional PID controller diagram is depicted in Figure 33.

As shown in the figure, a PID controller is comprised of three equations based on the error signal: proportional, integral, and derivative. Proportional control provides direct linear response to the current error present in the feedback. Integral control removes the steady-state error present in the system. Finally, the derivative control responds to the error's rate of change in the system to decrease overshoot. The core PID equations are listed below for both continuous and discrete controllers.

Different applications can call for one, two, or all three parts of the PID controller. For many applications, only the PI parts suffice. This arrangement provides adequate proportional response to feedback error while removing steady-state error. However, some applications need extra protection against overshoot and require a full PID setup.

For the purposes of a helicopter control system, it is ideal (if not required) to use all three parts of the PID. The I reset term corrects slow orientation drift while the D term guarantees fast corrections that could otherwise prove disastrous.

Equation 11. Core PID Control

$$e(t) = r - x(t)$$

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt} + u_0$$

$$u(k) = K_p e(k) + K_i \sum_{i=0}^{k} e(i) + K_d \left[e(k) - e(k-1)\right] + u_0$$

$$u(k) = u(k-1) + K_p \left[e(k) - e(k-1)\right] + K_i e(k) + K_d \left[e(k) - 2e(k-1) + e(k-2)\right]$$

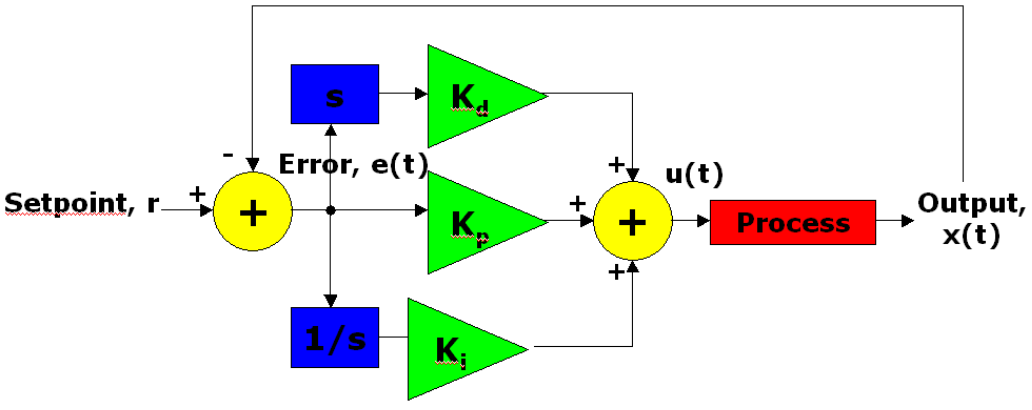


Figure 33. Traditional PID controller.

3.2.3 Position, Altitude, and Attitude Control

With knowledge of PID basics at hand, the UAV helicopter control system can be further examined. Here, the inclusion of the orientation estimate described in the previous section of this chapter acts as feedback to the PID loops. This section will focus on the PID control block comprised of three different loops: attitude, 2D position, and altitude.

The outer-loop guidance controller focuses on the translation of the helicopter from points A to B, or simply waypoint navigation. Thus, the destination waypoint determines the PID setpoint position. Additionally, because the desired flight behavior is a steady, stable hover between waypoints, the translation velocity should be minimized. Minimal velocity is achieved by setting the position-derivative setpoint to zero. The final guidance PID output is fed to the attitude controller for further use.

The inner-loop attitude control simply focuses on maintaining an attitude command or setpoint. For hovering, this setpoint is typically level (or slightly tilted to account for tail rotor thrust, fuselage imbalance, etc.). For waypoint navigation, the helicopter's attitude must be tilted in the direction of the desired heading. Thus, the guidance PID output is used as the attitude controller setpoint. In essence, the guidance controller outputs the approximate helicopter orientation needed to navigate to a destination waypoint, and the attitude controller maintains this orientation. Again, the helicopter should maintain a steady, stable hover. Therefore, the angular rate should be minimized by setting the orientation-derivative setpoint to zero. Finally, the generic helicopter commands (roll and pitch) are outputted by the attitude controller. Figure 34 depicts the guidance and attitude PID loops.

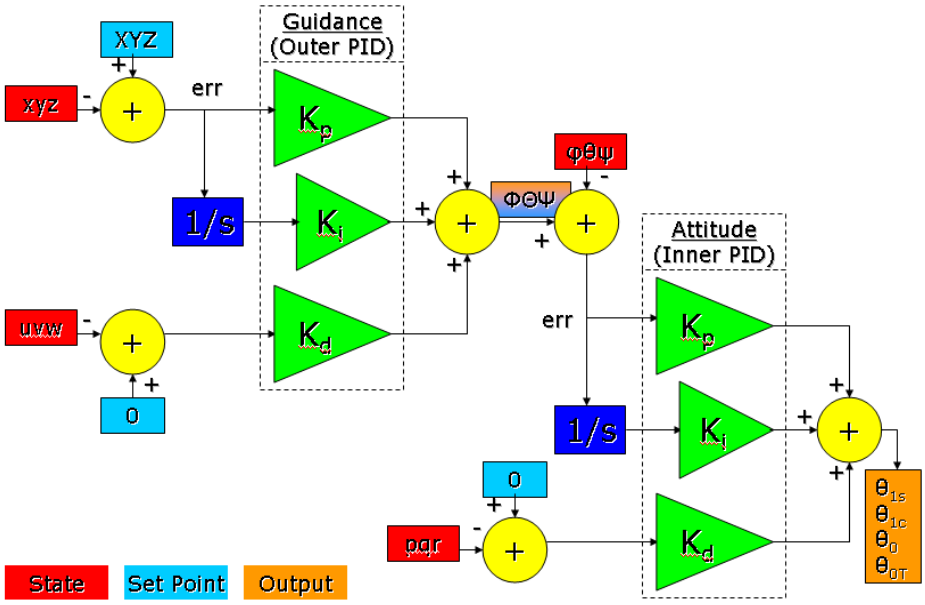


Figure 34. Two-step PID flight controller.

Controlling the altitude of the helicopter is just as important as controlling the attitude. A system hovering at perfectly-level orientation will still crash if the altitude is uncontrolled. Thus, a high-rate loop to monitor and control the altitude is a high priority for design. Unfortunately, the non-linear behavior of a helicopter's rotors – particularly when near the ground due to downwash – makes stable altitude control while adjusting attitude very difficult indeed. The design of an altitude controller follows the same simple PID design as that of the guidance controller: the desired altitude is used as the setpoint while the altitude-derivative is minimized. However, in this system, the altitude is manually-managed for maximum safety. Therefore, an altitude PID was not implemented.

The velocity form of the PID equations are used here for two reasons. First, the velocity form calculates the control as a change – preferable for a helicopter control system implementation. Secondly, the velocity form can rely entirely on orientation data. While the gyroscopes gives instantaneous change-in-orientation data, it can sometimes conflict with the much slower orientation data. The result is a much more finicky control system in practice. Therefore, by using only the constant (hopefully less noisy) orientation data, a better control system should result.

The basic attitude control system used onboard the test helicopter is shown in Figure 35 and the equations listed below.

Equation 12. Implemented PID Attitude Control using Orientation Estimate

$$e_\varphi(k) = r_\varphi - \varphi(k)$$

$$e_\theta(k) = r_\theta - \theta(k)$$

$$u_\varphi(k) = u_\varphi(k-1) + K_{p\varphi}\left[e_\varphi(k) - e_\varphi(k-1)\right] + K_{i_\varphi} e_\varphi(k) + K_{d\varphi}\left[e_\varphi(k) - 2e_\varphi(k-1) + e_\varphi(k-2)\right]$$

$$u_\theta(k) = u_\theta(k-1) + K_{p\theta}\left[e_\theta(k) - e_\theta(k-1)\right] + K_{i_\theta} e_\theta(k) + K_{d_\theta}\left[e_\theta(k) - 2e_\theta(k-1) + e_\theta(k-2)\right]$$

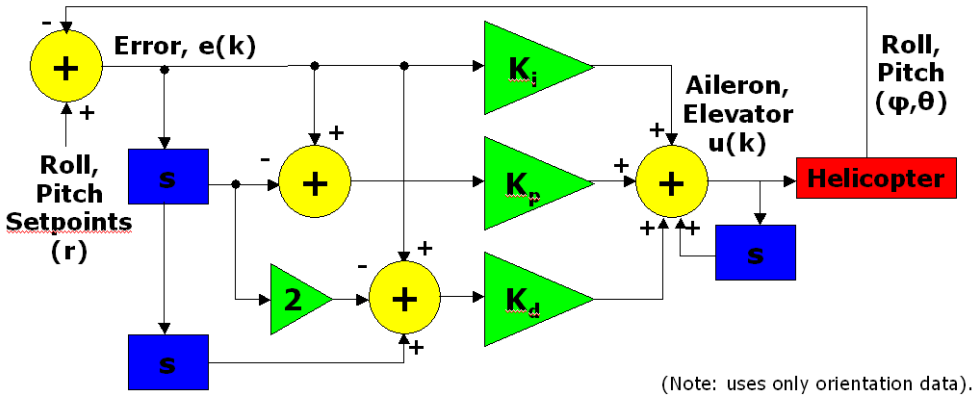


Figure 35. Simple PID Control System for Helicopter Attitude Control.

3.2.4 PID Tuning Techniques

Tuning the helicopter control to behave stably in any feasible system state is of paramount importance. Thus, the PID control must provide adequate speed of response to correct orientation before the system slips into an unrecoverable state. Additionally, the control must prevent the system from entering a state of oscillation (particularly increasingly unstable oscillation). This unstable state can be caused by too much overshoot by the control, oscillating around the set point in ever increasing magnitude. In this case, the helicopter will almost certainly crash very quickly.

A method of modeling the helicopter (plant) is required for accurate tuning of the PID loops for attitude and position control. Unfortunately, such a model is difficult to estimate without knowledge of the intrinsic parameters of the helicopter, such as moments of inertia, blade characteristics, aerodynamics of the fuselage and rotors, servo dynamics, etc.

Other methods are available for improving the real-world operation of a PID control system, such as bounding and dead-zones. Bounding is a non-linear operation employed by nearly every PID system to prevent transition into an unsafe control zone. For the UAV helicopter, bounding provides a finer amount of control on how much the system is allowed to correct itself at each time step. Ideally, this bounding will prevent unnecessary oscillation and increase stability; however, these advantages come at the cost of slowing the system response.

A dead-zone is not always used for PID systems, but can be an effective technique for minimizing control changes and stabilizing the system response. Essentially, a dead-zone prevents control changes below a certain threshold. Small, probably unnecessary, changes are filtered-out and the system is left in its current control state. For the UAV helicopter, most orientation close to level provide adequately for hovering in place. Thus, a dead-zone can be created a few degrees around level for better stabilization; however, this method again comes at the cost of slower system response.

While in a stable hover, the helicopter's orientation might not be perfectly level as expected (i.e., roll and pitch not zero). This roll/pitch error can be caused by a few effects, such as imperfect IMU mounting relative to the body frame, constantly-applied roll angle to compensate for tail rotor force, imbalanced center of gravity due to equipment mounting, etc. In any case, the PID loops must compensate for these effects in order to operate efficiently.

Near-optimal, "good" setpoints can be found using empirical data from manually-controlled flights. The orientation data is first recorded during several phases of stable hovers. Afterwards, the data files are parsed and processed statistically. The mean roll/pitch values of each hover session give estimates of the optimal setpoints. The variances and standard deviations of the hover sessions give an estimate of the setpoint quality. The weighted-mean of these setpoints can then be found, giving "good" roll/pitch values for a stable hover.

3.2.5 Controls/Servo Mixing

When flying a standard R/C helicopter, the transmitter itself handles all of the mixing to translate the four stick commands (elevator, aileron, throttle, and yaw) into

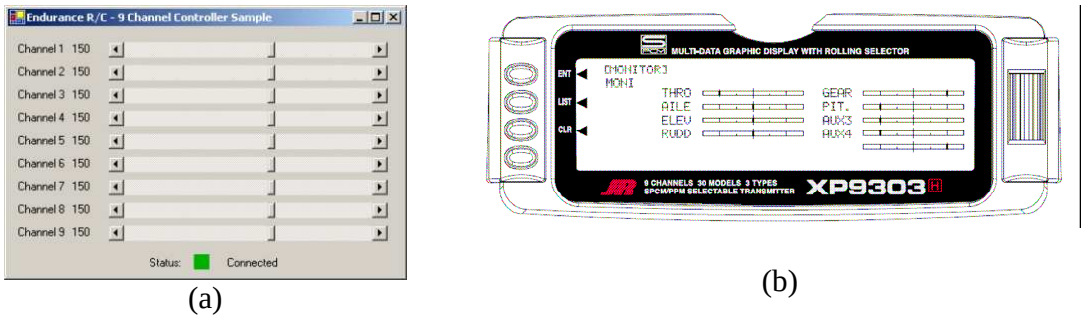
servo commands. Unfortunately, much of this had to be reengineered in order to move the control system to a PC – regardless of which servo control scheme was used. Three types of mixing were considered when re-engineering the control system: (1) throttle/collective-pitch mixing, (2) aileron mixing, and (3) elevator mixing.

The data for mixing was obtained primarily through empirical data in order to obtain the exact values used by the standard R/C hardware for the IRIS helicopter. Once obtained, piece-wise linear approximations could be easily made and used for on-the-fly control/servo mixing.

During flight-testing of the automatic PC control system, the model repeatedly exhibited undesirable response (i.e., pitching hard backwards when the model was close to level). Initially, this bad behavior was chalked-up to poor IMU data. However, even as the IMU accuracy increased and reported closer-to-correct orientations, the control system continued to incorrectly pitch the helicopter. To troubleshoot this behavior, the system response was re-evaluated using static tilt testing.

The static tilt tests seemed to work fine when the PC control system was given full control over the servos. Yet, repeating the test while limiting the PC control to only Aileron/Elevator through the transmitter gave different results. Namely, the swash-plate response was much lower relative to the tilt angle, and some orientations seemed to perform better than others. Initially efforts to correct the response focused on the PID loop. The allowable PC control servo throws were increased, and the PID gains were all incrementally increased. However, the flight-testing continued to fail with all settings. Thus, the transmitter behavior while limiting slave control was further investigated by examining input data from the slave PC and output data from the R/C transmitter to the servos. (Note: the output data from the transmitter is essentially “raw” servo data. The R/C receiver merely decodes the PPM signal and sends it directly to the servos). Figure 36 depicts the interfaces used to check the transmitter data from the PCTx.

Through experimentation, it was shown that the XP9303 transmitter in master mode simply transmits the PPM signal directly from the trainer port while the trainer switch is held. Thus, the slave must calculate all CCPM, trims, etc. and forward the positions to the master. This behavior called for the CCPM and trim settings to be reverse-engineered from the transmitter inputs/outputs. Afterwards, all input positions were observed to produce identical outputs to the transmitter.



(a) (b)

Figure 36. Trouble-shooting the "Double Servo Mixing:" (a) PCTx control applet, (b) XP9303 Monitor .

However, during flight-testing, the slave is only given limited control over the model for maximum safety. The master R/C transmitter, controlled by a human pilot, has full control over the Throttle and Rudder at all times. Control of the Aileron and Elevator can be given to the slave by holding the trainer switch on the transmitter. Unfortunately, all flight tests with limited slave control seemed to fail. Initial observations seemed to implicate the spotty IMU performance. However, further tests showed incorrect control system response even when the IMU seemed to be performing stably. Furthermore, the static tilt tests showed significantly different control system response between full and limited slave control.

After some initial observations of the slave and master output data, the author discovered that the XP9303 transmitter assumes different input schemes on the trainer port when using full or limited slave control. When limiting the slave's control on the XP9303, the master uses its programmed CCPM and trim settings. Therefore, the slave is only responsible for sending generic throttle, rudder, aileron, and elevator commands encoded as PPM. Essentially, these are non-CCPM helicopter servo positions for a zero-trimmed model.

3.2.6 Flight Data Collection

The flight data collection system sought to collect both inertial and control data during carefully controlled manual flights for later analysis. By analyzing the real data, it was hoped to achieve a better-tuned control system. Many sessions of data were captured. Unfortunately, many of the first data batches suffered from the severe IMU drift that also plagued the Kalman filtering experiments.

The data collection system consisted of three primary parts: the onboard laptop PC, the Xsens inertial measurement unit (IMU), and the FMA Co-Pilot receiver. The laptop logged the helicopter orientation, calibrated sensor values, raw sensor values, transmitter controls sent, servo positions, and infrared-estimated orientation.

The FMA Co-Pilot receiver outputs current state data via an external RS232 connection on the "T" pins. The FMA "FS Viewer" software reads this data from the PC's COM port in real-time and records it to an easily parsable text file. For these experiments, only the logged servo position data was considered.

The Xsens IMU outputs estimated orientation (quaternions, Euler angles, or Direct Cosine Matrix), calibrated sensor values, and raw binary sensor data. The internal sensors include gyroscopes, accelerometers, magnetometers, and a temperature sensor.

3.2.7 Control System Hardware Implementation

The PC control system combines the hardware with the algorithms. The system is the real-world implementation of the helicopter control system using a standard PC and COTS parts.

The initial control system discards much of the original R/C hardware in favor of direct servo control. In order to maintain a failsafe in the event of software failure, the throttle (channel 1) and tail control (channels 4/5) are always controlled by the human

pilot through the standard transmitter. However, the blade cyclic and collective pitches are both controlled directly by the PC.

The Co-Pilot receiver can log both infrared orientation and control data in real-time, transmitting the data over standard serial RS-232. Thus, flight data can be saved and examined in later experiements.

The Pololu 16-servo controller is a USB or UART device capable of generating sixteen individual PPM signals on sixteen sets of three-pin connectors. When using the supplied driver, the Pololu appears as a serial COM device in Windows XP. Data can be sent to the Pololu by simply writing to its COM port in any programming language.

The Endurance R/C PCTx adds more flexibility and safety to the original control system. The necessity of keeping all hardware onboard is removed by pushing all control data directly through the R/C transmitter. Furthermore, the pilot can instantly transfer control between himself and the PC at any time. Thus, a software failure should (theoretically) never lead to a crash. Additionally, the pilot can raise the helicopter to a safe altitude before control is transferred to the PC and transfer control instantly back in the event of a flight emergency.

Most standard R/C transmitters have a Trainer port and switch for instructing new pilots. When used in the normal fashion, the instructor's transmitter is setup as the Master and the student's as the Slave. Holding the Trainer switch on the Master transmitter allows the student limited or full control of model. On the JR9309 transmitter, the student can control any or all of the (1) Throttle, (2) Aileron, (3) Elevator, and (4) Rudder. For most experiments while tethered, the student, in this case the PC, will control only the Aileron and Elevator.

The data sent from the Slave transmitter to the Master is simply encoded as standard PPM. When transmitting multiple channels on a PPM signal, a long sync pulse is sent followed by a HIGH pulse for each channel separated by (usually) fixed-length LOW intervals. The standard transmit rate is 50-Hz, or a full frame update every 20-ms. The individual channel pulse lengths contain the servo position, where 1500-ms denotes neutral, 1000-ms a 45° offset in one direction, and 2000ms a 45-° offset in the other.

Since only standard PPM is being sent, the Slave signal can be emulated using a microcontroller with a timer module. However, for ease of implementation, the pre-programmed Endurance R/C PCTx was purchased to remove the cost of developing the hardware in-house.

The PCTx connects to the P/C via a USB port, and appears as a Human Interface Device (HID) in Windows XP. Simple C# and C++ APIs are included with the PCTx for integration into custom projects. The PCTx outputs a PPM signal on a Mono phone plug for connection to the R/C transmitter.

Software communication with the PCTx is more primitive than with the Pololu servo controller. Only ten bytes total are sent per frame: one sync ZERO byte and nine bytes of channel data. The Pololu, on the other hand, requires a minimum of five bytes per servo for seven bit resolution and six bytes for fourteen bit (the most-significant bit of every non-sync byte must always be zero). The additional bytes allow for better out-of-sync error catching and servo resolution. However, the PC (in these experiments) controls only three of the nine channels. Thus, the data transmission overhead of the Pololu is only 80% more (one eighteen byte packet versus one ten byte packet per frame).

Therefore, the PCTx is at the disadvantage of having significantly lower servo resolution. Through experimentation, the PCTx was found to have a servo resolution of roughly 8-ms, compared to the Pololu's 0.5-ms. However, for most applications this should serve well enough.

The software algorithm for collecting orientation data and outputting servo commands can be summarized in six steps:

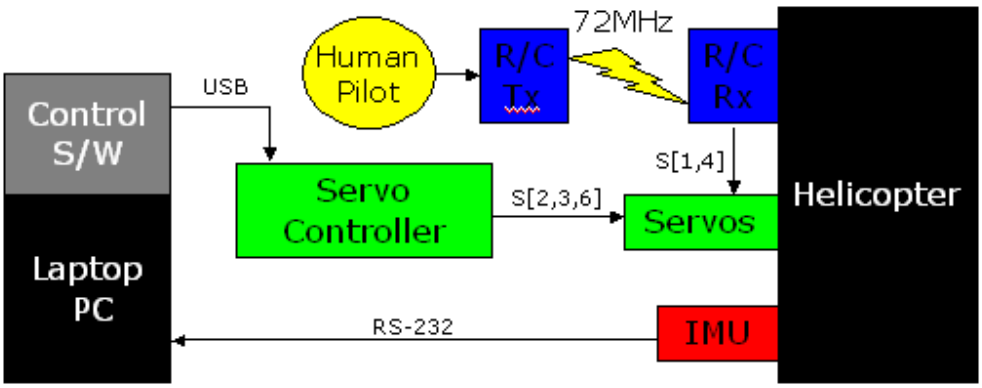
1) Poll IMU for new calibrated sensor and orientation data. If new data found, go to Step 2. Else, continue polling.
2) Send orientation estimate to attitude-PID loop and process. Gains and set-points are pre-determined and static. Output compensating Elevator and Aileron commands to hover the helicopter. If PC has limited control (Aileron/Elevator only), go to Step 4.
3) Calculate swash-plate servo mixing using piece-wise formulas.
4) Convert and round aileron/elevator commands (or servo-mixes) to integer format for the servo controller. For the Pololu, the proper range lies between 500-5000 (0.25-sec to 2.5-sec). For the PCTx, the proper range lies between 73-217 (0.45-sec to 2.1-sec).
5) Send the servo controller values to the manufacturer's C++ serial/HID-wrapper module.
6) For the PCTx, hold the Trainer switch to allow PC control. The Pololu configuration does not allow manual control through the transmitter.

Finally, the two hardware setups based on different servo control methods described in this section are summarized in the following figure. Figure 37 (a) depicts the Pololu-based hardware setup and Figure 37 (b) depicts the version based on the Endurance R/C PCTx.

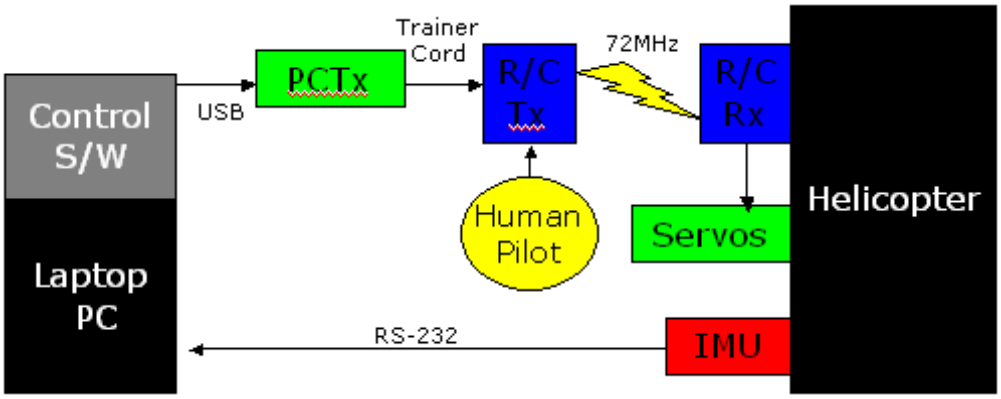
3.3 Summary and Conclusions

This chapter has described the theory of a sensor/interceptor placement planning system and an accompanying UAV helicopter flight control system (FCS). These components are part of a automatic wide-area surveillance system utilizing perimeter camera emplacements for intruder detection. In the complete system, a detected intruder by the perimeter cameras signals a helicopter to investigate at that location. Thus, the sensor placement planning calculates the shortest-path to that intruder's estimated location and uploads the waypoint path to the helicopter-interceptor. The UAV helicopter then navigates to the initial location, attempts to detect and track the intruder, and generates and navigates to waypoints for following until the intruder is apprehended.

Here, only the hovering sub-system is implemented for the UAV helicopter. Further work to extend the control system to waypoint navigation was beyond the scope of this thesis.The next section will cover the experiments conducted involving the UAV helicopter flight control system.



(a)



(b)

Figure 37. Old and new hardware setups: (a) Pololu 16-servo controller-based hardware setup and (b) Endurance R/C PCTx-based hardware setup.

4. Experiments

The overall wide-area surveillance system can be broken into several different components, each with individual experiments. This section will describe those experiments and corresponding results. First, the sensor placement planning experiments are detailed, followed by helicopter flight control system experiments. Within sensor placement planning, individual experiments for optimal camera placement planning, shortest-path interceptor placement planning, and automatic, video-based object/face recognition and tracking are covered. The flight control system experiments include controls/servo mixing, improving the IMU orientation estimate, and finally full flight testing.

4.1 Sensor Placement Planning Experiments

These rounds of experiments dealt with the automatic monitoring of a given area defined in 2-D by simple polygons. First, the optimal camera placements are found to maximize total coverage while meeting cost constraints. Next, the shortest-path placement can be found for the interceptor helicopter. This resulting location is the "home base" of the helicopter within a room to minimize time-to-intercept of any detected intruder. Finally, a proof-of-concept experiment for automatic object detection and tracking is demonstrated. In this experiment, an intruder is recognized and tracked via video and a filtered location estimate is the output.

4.1.1 Optimal Camera Placement Planning

The sensor planning experiments begin with an area to be monitored. This area is defined by a simple polygon perimeter and any number of simple polygon holes within the perimeter. Additionally, the user inputs a set of solution constraints as defined in previous sections.

The experiments were performed in MATLAB. To find each of the following results plots, several processing steps must be completed. First, a set of "good" test points are generated on the boundaries of the test area. Next, the visibility polygons are calculated for each of these test points. A grid overlay is put over the test area, and the visibility polygons are translated into how much of the grid they encompass. Multiple polygons can then be easily combined with simple AND/OR operations.

Next, the camera modeling is performed. Each available camera is tested at each test point in up to three orientations (centered and along each attached edge). PTZ cameras are given optimal field of view and panning parameters to maximize coverage. The resulting mask set is then fed into a branch and bound algorithm with constraints, and the optimal mask set can be found from there.

Four plots are generated to demonstrate the sensor planning working on a simple room. First, the minimum field of view for recognition requirements is plotted for each potential camera placement within the room. The 'hottest' areas near the cameras allow for recognition even with a field of view up to π or 180°. Secondly, the total camera coverage plot shows the amount of overlap present at each point within the room. Camera overlap provides redundancy and theoretically increases the chance of detection, but also

increases the overall cost and decreases placement efficiency. Thirdly, the solution coverage that meets input requirements is plotted. This camera placement maximizes coverage while minimizing cost. Finally, the overlap in the solution is plotted to analyze placement redundancy. Four test areas are demonstrated here based on real-world environments.

All experiments are performed using a real-world derived set of test cameras. These test cameras are allowed a few parameters relevant to sensor placement planning: frame resolution in (pixels), PTZ (true/false), cost ($), field of view (min/max in degrees), and pan/tilt range (max in degrees). The first camera was fixed, cost $600, and had min/max fields of view of 27° and 67°, respectively. The second camera was a PTZ, cost $600, had min/max fields of view of 44° and 140°, respectively, and a max pan range of 140°. The third camera was fixed, cost $900, and had min/max fields of view of 36° and 75°, respectively. Finally, the last camera was a PTZ, cost $1500, had min/max fields of view of 1.73° and 55.8°, respectively, and max pan range of 360°. All cameras used here had 640-pixel horizontal resolution image frames.

This first experiment (Figure 38) is based around an actual room within Ferris Hall on the UTK campus where many IRIS students work. The area is modeled empty as a very simple polygon that could be adequately covered with a single camera.Here, the search constraints include a minimum coverage of 90% at a minimized cost. The results place a single $600 fixed camera at (-15, 10) with a 67° field of view and a rotation of -5°. The objection detection coverage is 92.5% at spatial recognition of 5 pixels/face.

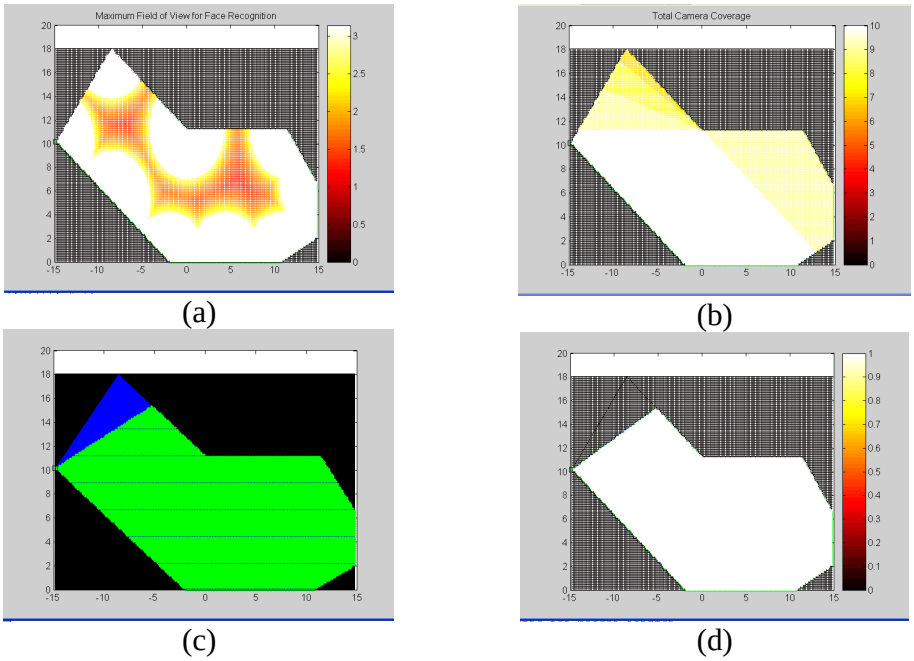


Figure 38. Ferris Hall, Room 209. (a) Minimum FoV for SR Requirements, (b) Visibility Polygons Coverage, (c) Solution Camera Coverage, and (d) Solution Camera Overlap.

The second experiment uses the IRIS West indoor helicopter testing area. This area has a car lift in the Southwest corner (denoted by a hole), and two cars parked against the opposite wall. Figure 39 includes a few pictures of the area. The car lift pictured in Figure 39(a) is denoted by the small hole in the lower left of the area. The cars in the far rear of Figure 39(c) are denoted by the concave peninsula in the lower right.

This area is the primary testing facility for the helicopter interceptor experiments detailed later in this chapter. This step seeks to find optimal camera placements within the area towards a complete two-step surveillance system. Moreover, the area provides a real-world testing center for the helicopter in conjunction with object/face recognition and tracking.

The results for this experiment use three camera placements: one $600 fixed and two $600 PTZ cameras. The first fixed camera at (15, 115) has a 90° field of view and rotation of -45°. The second PTZ camera at (15, 40) has a 140° field of view, a rotation of 45°, and a 130° pan range. The third PTZ camera at (15, 20) has a 140° field of view, -45° rotation, and 130° pan range. The total detection coverage is 93.88% at minimum spatial recognition of 5 pixels/face with 30.98% camera overlap.

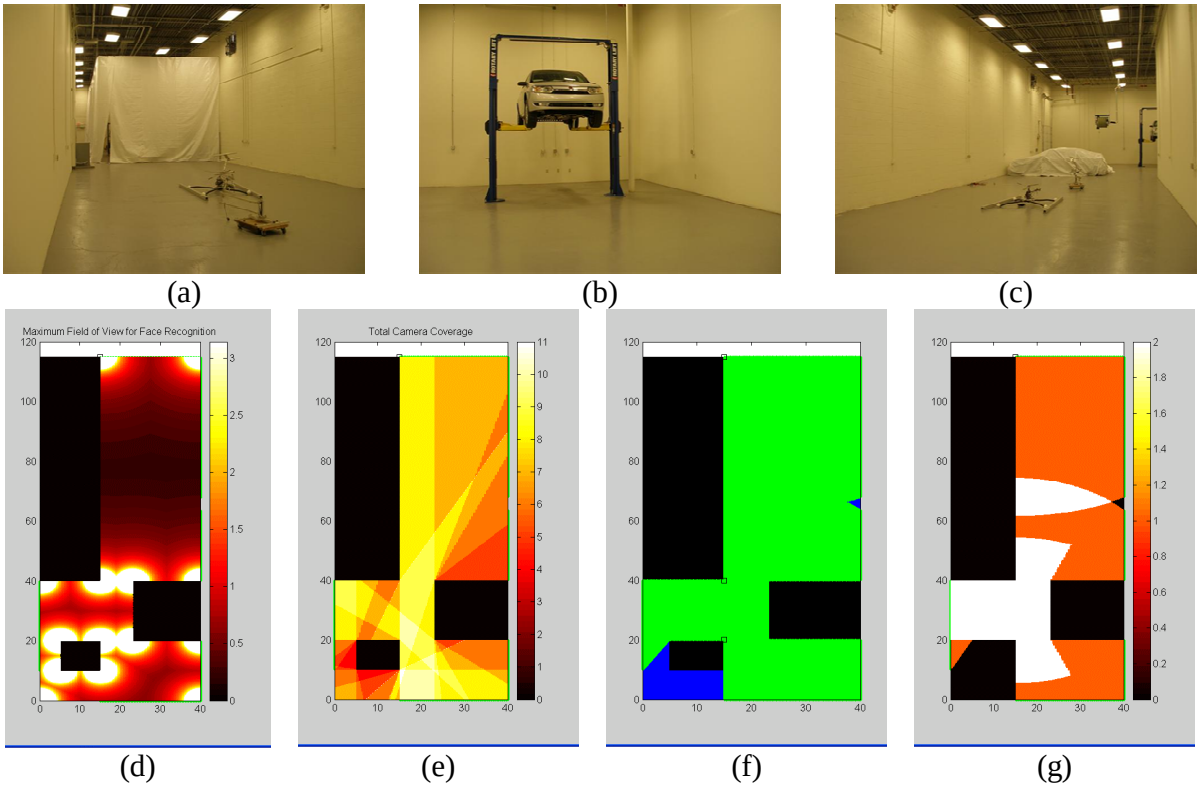


Figure 39. IRIS West Indoor Helicopter Testing Area: (a-c) snapshots; (d) Minimum FoV for SR Requirements; (e) Visibility Polygons Coverage; (f) Solution Camera Coverage; and (g) Solution Camera Overlap.

The third experiment used a theoretical typical room layout comprised off one center pillar hole and a half-wall. The area was derived from a sample figure used in [Erdem and Sclaroff, 2005]. Therefore, this area provides a solid test of the Radial Sweep algorithm implementation in conjunction with the camera modeling process. As can be seen in Figure 40, twenty-four test points are generated along the perimeter of the area. While many of these points cover a large portion of the area, the half-wall provides an interesting problem of occluding most of the lower-right section. Furthermore, placing a camera in this lower-right section provides only a small increase in coverage compared to other locations.

To achieve >90% coverage, two camera placements comprised the solution set. The first $600 fixed camera at (0, 0) has a 67° field of view and 45° rotation. The second $600 PTZ cameras at (20, 0) has a 44° field of view, 90° rotation, and 136° pan range. The total detection coverage is 92.6% at minimum spatial recognition 5 pixels/face with 42.3% camera overlap.

These results achieve the coverage goal, but also possess a high amount of overlap. While overlap provides redundancy and the potential for reliable camera hand-off (passing an intruder between cameras), it is also wasteful in terms of coverage potential. Unfortunately, in order to meet minimum spatial resolution requirements for detection, the field of view was kept narrower than maximum available. This restriction led to the addition of the second camera (placed in the lower-center) that covers a small portion of the lower-right of the area and overlaps much of the central area with the first camera.

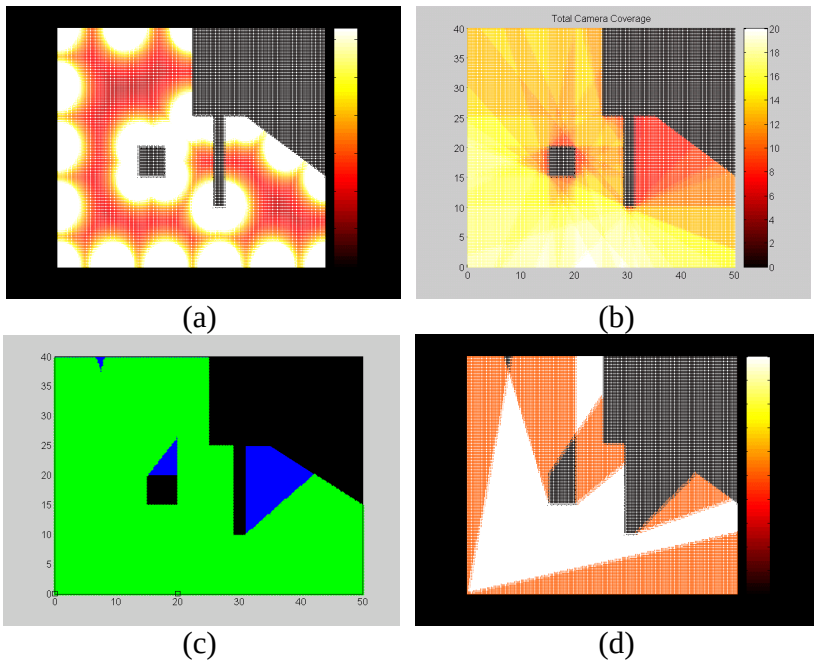


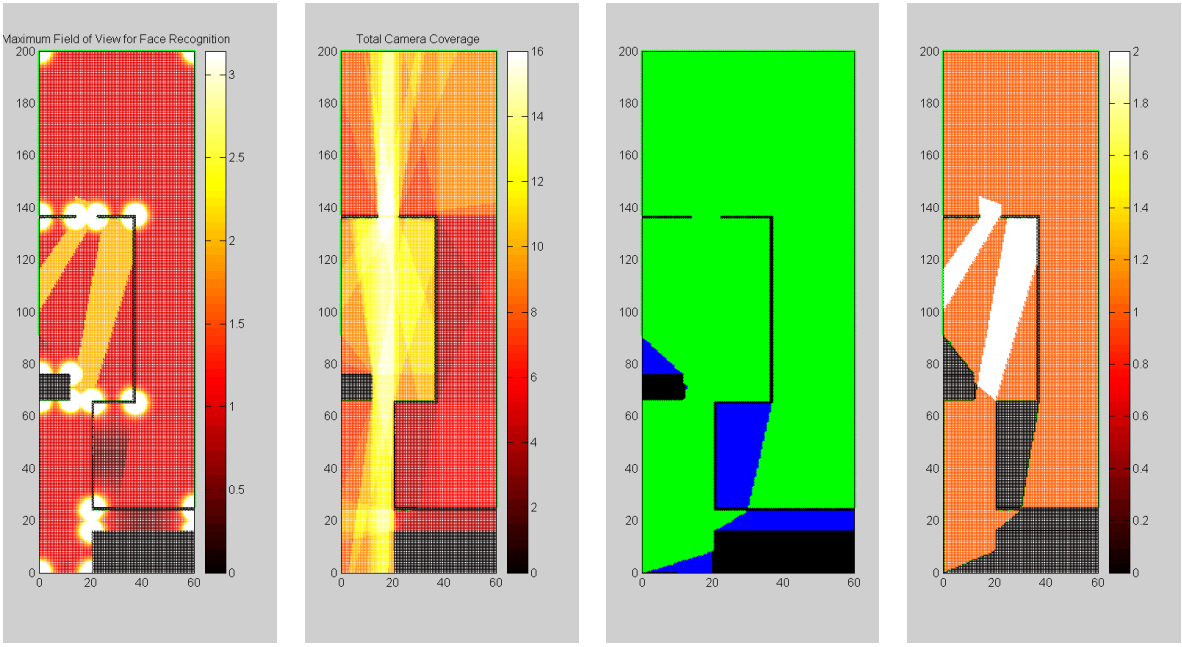
Figure 40. Typical Room Layout 1. (a) Minimum FoV for SR Requirements, (b) Visibility Polygons Coverage, (c) Solution Camera Coverage, and (d) Solution Camera Overlap.

The final experiment dealt with the entire IRIS West facility. The layout can be seen in the floor plans shown in Figure 41. The solution includes three camera placements totaling $1,800. The first $600 fixed camera at (0, 0) has a 67° field of view and 57° rotation. The second $600 fixed camera at (20, 60) has a 140° field of view and 70° rotation. Finally, the third $600 fixed camera at (60, 200) has a 90° field of view and 225° rotation. Actually, these three cameras are actually PTZ cameras modeled as fixed. The total detection coverage is 90.95% without minimum spatial recognition requirements and 9.79% camera overlap. Figure 41 depicts this experiment.

4.1.2 Shortest-Path Interceptor Placement Planning

The next round of experiments focused on optimal interceptor placement. In essence, the same rooms from the previous sensor planning experiments are re-examined to find optimal placement of a UAV helicopter interceptor within the perimeter. Thus, when combined with the placed cameras, a complete surveillance-interception system is created. Figure 42 depicts these experiments.

First, the theoretical typical room layout was processed. This room is a rather ordinary are with a center island pillar and a half-wall. Here, each interceptor is given a maximum radius of 20 units. This maximum radius is due to a time constraint (maximum range within some time T). All distances are calculated using shortest-paths around obstacles. Coverage >90% was achieved using two interceptors for this area.



(a) (b) (c) (d)

Figure 41. IRIS West Full. (a) Minimum FoV for SR Requirements, (b) Visibility Polygons Coverage, (c) Solution Camera Coverage, and (d) Solution Camera Overlap.

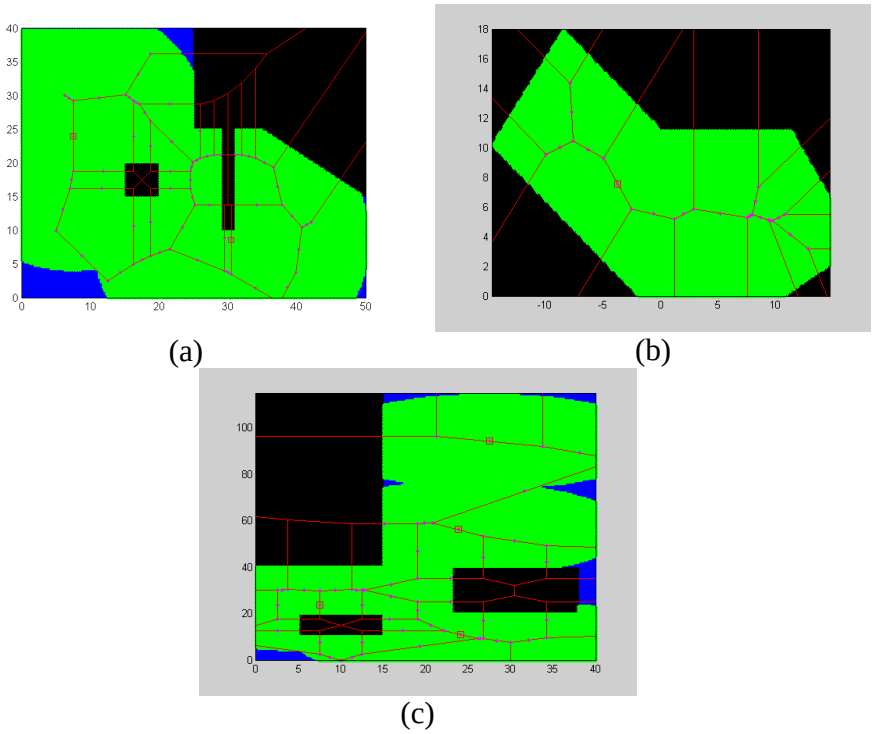


Figure 42. Voronoi Diagram (red lines), Sampled Test Points (pink circles), and Final Interceptor Placements (red squares) for (a) Typical Room Layout #1; and (b) Ferris Hall, Room 209; and (c) IRIS West Indoor Helicopter Testing Area.

The next experiment uses the simple Ferris Hall Room 209 layout. Again, each interceptor is given a maximum radius of 20 units. The simple solution uses one interceptor placed along the center of the Voronoi skeleton. It's interesting to note that the centroid point of this room has 100% visibility of the area – making it the optimal solution for this particular layout. However, the chosen solution based on Voronoi skeleton and shortest-patch mask searching also yields 100% coverage.

Third, the IRIS West Indoor Helicopter Testing Area is processed for optimal interceptor placement. Each interceptor was given a maximum radius of 20. However, this layout required four cameras to provide >90% coverage within the radius (time) constraint. The vertical scale is much larger than the horizontal, giving this plot an unfortunate distorted effect. For example, the lower-left hole is actually a square.

4.1.3 Object/Face Detection & Localization

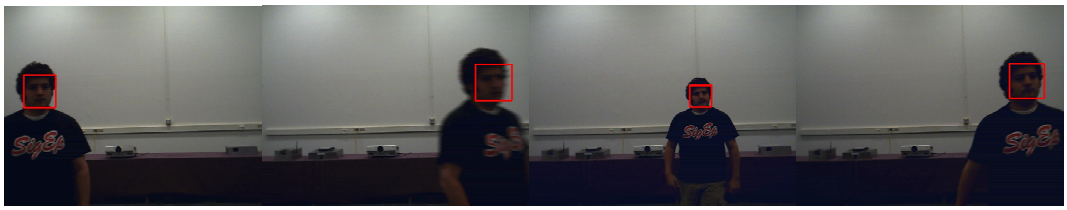
These experiments were conducted as part of the sensor placement planning for persistent surveillance using perimeter camera emplacements; thus, the data is captured from the viewpoint of a perimeter, fixed video camera. These proof-of-concept experiments use inexpensive web-cameras in conjunction with free, open-source imaging software (Intel OpenCV) to accomplish the goal of automatic intruder detection and localization.

Unfortunately, face detection (even the relatively optimized methods used here) requires a great deal of processor power. The fastest monitoring rate attained was around 4-Hz, using 250-ms processing time per frame. Figure 43 (a) includes some processed frames captured during a typical experiment.

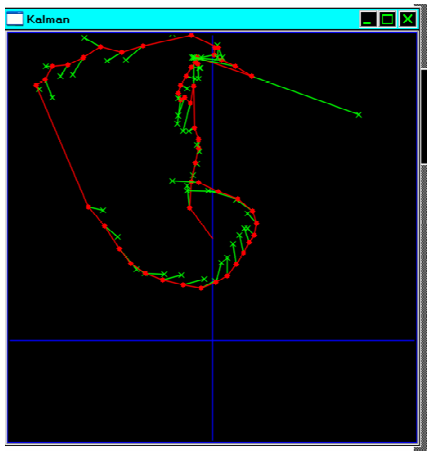
Because the face detection uses so much processing power, including full camera calibration for an optimal localization estimate was not practical. However, future work on this project should include experiments using calibrated cameras. For now, a simple geometry-based algorithm estimates the location of a detected object/face within the scene using only the known field-of-view of the camera lens. The algorithm then yields the polar coordinates of the object/face relative to the camera.

Since the object/face localization needs only be a rough estimate – a general search area for the UAV helicopter to intercept – the simple algorithm should suffice. However, there are circumstances where very bad estimates will occur. Fortunately, a simple Kalman filter on the position and velocity states of the intruder should improve performance significantly. Figure 43 (b) shows the typical output with measured data, Kalman predicted data, and Kalman updated data. Figure 43 (c) shows how this data can be Kalman smoothed, forward- and reverse-Kalman filtered, for even better results.

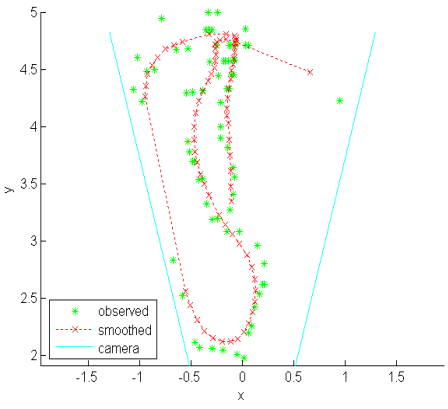
The overall face detection and localization results were encouraging towards further experimentation. Future work would focus on improving the initial estimate.



(a)



(b)



(c)

Figure 43. Face detection and localization demo: (a) snapshots, (b) Real-time measured localization estimate (green) versus Kalman filtered (red), and (c) Kalman-smoothed localization estimate.

4.2 Flight Control System Experiments

4.2.1 Servo Mixing

One of the first steps in the design of the PC control system involved the reverse-engineering of the transmitter servo-mixing. The original design called for direct control of the helicopter's servos through a generic USB servo controller. Later, a custom interface unit was purchased that allowed commands to be sent from the PC to the R/C transmitter directly. For testing purposes, this new setup proved optimal. However, for either case, the PC needed to output raw servo positions. Thus, after calculating the desired aileron and elevator commands, the PC also had to calculate the actual mixed servo positions.

The helicopter's servo mixing is based on Cyclic-Collective Pitch Mixing (CCPM). However, the exact pitch curves used by the transmitter were custom-tuned for the IRIS helicopter. Therefore, the servo mixing was reverse-engineered from the transmitter input/outputs. To accomplish this task, the manual inputs were set at each minimum/maximum position and the receiver position-pulse modulation (PPM) output was recorded. The collected data is summarized in Table 5 and Figure 44.

While this data could instead be derived from the trim and pitch curve settings stored within the transmitter, the above method allowed for exact knowledge of servo positions for each input combination. With the neutral and min/max positions for each servo, simple formulas were derived for the calculation of servo mixing given any throttle, aileron, and elevator input. These formulas are presented below.

Equation 13. Servo-Mixing

$$ch1 = \begin{cases} 900 + 20.2 \cdot thr, thr \le 50 \\ 1910, thr > 50 \end{cases}$$

$$pitch\_mix = \begin{cases} 3.7 \cdot thr, thr \le 50 \\ 5.36 * thr - 38, thr > 50 \end{cases}$$

$$ail\_mix = 4.8 \cdot ail - 240$$

$$elev\_mix = 2.4 \cdot elev - 120$$

$$ch2 = 1702 - pitch\_mix - elev\_mix - ail\_mix$$

$$ch3 = 1664 - pitch\_mix + 2 \cdot elev\_mix$$

$$ch6 = 1416 + pitch\_mix + elev\_mix - ail\_mix$$

Initial flight testing began with all equipment mounted within the safety gear's cage attached below the helicopter. The hardware included the Xsens MT9-B IMU, Pololu 16-servo controller controlling swash-plate servos (2,3,6), a laptop processing inertial data and generating PID-based controls, and a servo controller. While testing, the pilot had full control of the throttle (motor speed) and the tail servo.

Table 5. Reverse-Engineering the Transmitter Servo Mixing

Inputs	Channels				Change from Neutral		
	1	2	3	6	2 (Adj)	3 (Adj)	6 (Adj)
Center	900	1700	1663	1414			
Left	900	1945	1662	1655	*243*	*-2*	*239*
Center	900	1703	1663	1417			
Right	900	1466	1663	1177	*-236*	*-1*	*-239*
Center	900	1702	1663	1414			
Down	900	1823	1423	1297	*121*	*-241*	*-119*
Center	900	1704	1663	1416			
Up	900	1582	1904	1536	*-120*	*240*	*120*
Center	900	1702	1665	1417			
Up-Left	900	1820	1906	1777	*118*	*242*	*361*
Center	900	1703	1664	1417			
Down-Left	900	2066	1420	1537	*364*	*-244*	*121*
Center	900	1703	1663	1415			
Down-Right	900	1584	1419	1050	*-118*	*-245*	*-366*
Center	900	1701	1662	1413			
Up-Right	900	1340	1906	1296	*-362*	*242*	*-120*
Center	900	1702	1666	1417			
Half-Throttle	1910	1516	1476	1599	*-186*	*-188*	*183*
Full-Throttle	1910	1249	1208	1867	*-453*	*-456*	*451*
Center	900	1702	1664	1416			

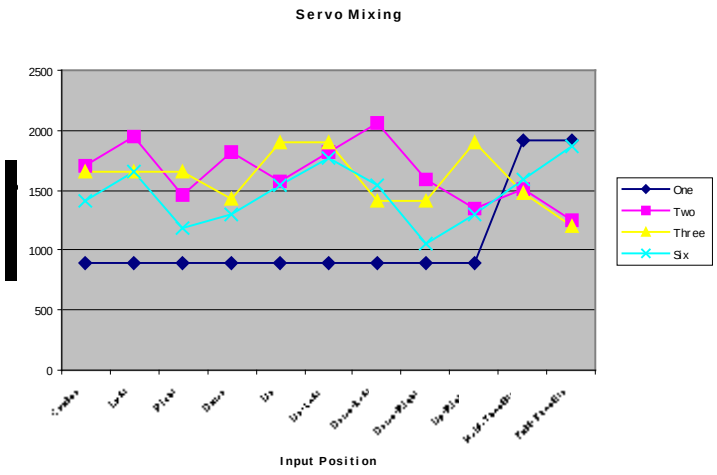


Figure 44. Servo Mixing Receiver Outputs at Each Input Position.

During flight testing, the computer controls the swash-plate completely, including the ability to change the collective pitch (which determines how much force the blades push downwards). The collective pitch was fixed at the 50% throttle setting. Here, the helicopter pulls hard-left the entire time the blade is spinning, making it unsafe to leave the ground. Only video data was used for verification and inertial/control data was not logged.

The PID gains were tweaked and a dead-zone was implemented to attempt to fix the unnecessary hard-left compensation. Again, the helicopter repeated the same tendency to pull hard left when the blades were spinning at full speed. Only video data was used for verification and inertial/control data was not logged.

Testing then moved to a new hardware setup using the purchased Endurance R/C PCTx. Most of the hardware can be moved offboard while testing using this configuration. The Pololu servo controller is replaced with the PCTx, allowing all commands to be sent through the main R/C transmitter. The Xsens IMU is connected directly to the PC over a long RS-232 serial cable. The cabling runs down the length of the safety gear. The battery for the Xsens IMU is also removed and an AC-DC power supply is used instead. The laptop sits beside the transmitter, connected by the USB PCTx. Here, the PC only directly controls the Aileron and Elevator.

(A note here: the PCTx invariably locks up if the transmitter antenna is extended. There seems to be no workaround. The antenna was damaged in the helicopter crash and will eventually need replacement for outdoor flights with the PCTx).

An attempted flight was made using the new hardware. The vibrations of the helicopter shake the receiver battery free from the canopy near the end of the recorded flight video! (Just before liftoff under manual control). Very luckily, the motor controller automatically powers down the motors when it loses the signal from the receiver.

A second flight attempt was made with the new hardware. The helicopter is manually flown until stable on its skids. The Trainer switch is then held briefly, allowing the PC to control the helicopter. The helicopter immediately bucks hard left again, and manual control is resumed while the rotor spins down. Automatic control seems to be failing. The previous and current flight data must be analyzed to find the problem.

A recurring problem with IMU data was found and documented on video. The laptop display is recorded on the left side of the video displaying a wire-frame of the IMU orientation in real-time. The right side of the video shows the helicopter while being manually flown. As can be clearly seen on the wireframe, the IMU drifts a bit as soon as the rotors start to spin up. Some of the heading error could be due to the magnetic field generated by the motor. As the motor reaches full speed, the wire-frame spins nearly completely upside down, while the helicopter is clearly level to the ground.

Initial conclusions were drawn from these rounds of testing and solution paths were theorized. First, the IMU data becomes near-useless under the full-throttle vibrations of the helicopter.

- Solution #1: Remount the IMU in a different location to decrease the vibration noise, move the IMU farther away from the magnetic field generated by the

motor, and also shield the IMU from the downwash generated by the main rotor blades.

- Solution #2: Attempt to tune some of the IMU settings to better cope with the extreme vibration noise.
    - One hypothesis is that the accelerometer data is being preferred over the rate-of-turn data in the sensor fusion algorithm. Since the helicopter is generating a large amount of force on itself while lifting the safety gear, the accelerometer data is skewed badly.
    - The magnetic interference caused by the motor might be corrupting the magnetometer readings, which might need to be weighted very small compared to the other two sensors (accelerometers and gyroscopes).
- Solution #3: Capture more flight data without the burden of the safety gear and while lifted far enough off the ground to eliminate downwash effects. This should give the best possible readings from the IMU.
- Solution #4: Rewrite the sensor fusion algorithm on calibrated sensor data (Kalman filter the IMU sensors). This strategy could generate better or worse results depending on how well the helicopter vibration noise is modeled. The main benefit is the addition of control data for the filter.

4.2.2 Improving the IMU Orientation Estimate

The "crossover" frequency of the MT9-B essentially determines what sensors are used with certain frequency data. High-frequency data will be preferred by the gyroscopes, while low-frequency data will be preferred by the accelerometers and magnetometers. The default frequency cut-off between these two is at 1.0-Hz. High-frequency data seen by the accelerometers is most likely noise, not actually forces on the IMU, and should be filtered.

One theory for the cause of the IMU error involved the upward thrusting force of the helicopter. Because of this constant force on the IMU while remaining rather motionless, it was postulated that the IMU might be mistaken some of this force for gravity. This situation could have been exacerbated by the straining of the helicopter against it tethers while lifting off. Figure 45 depicts an experiment where the MT9-B sensor crossover frequency was lowered to a minimal value (0.01 Hz), hopefully to eliminate the orientation error. Unfortunately, the error remained in this test.

A second theory proposed that EMI could be corrupting the readings of the magnetometers, and possibly the other sensors as well. The MT9-B offers the ability to weight the magnetometer readings against those of the accelerometers, and two more flight experiments were attempted while adjusting this weighting. First, the magnetometers were removed from the MT9-B sensor fusion and the tethered, powered flight test was repeated. (magnetometer weighting set to zero). As seen in Figure 46, the IMU orientation error occurred anyway. In fact, the sensor performance was much worse than before, and did not correct itself when the helicopter touched-down. This avenue of experimentation was quickly abandoned.

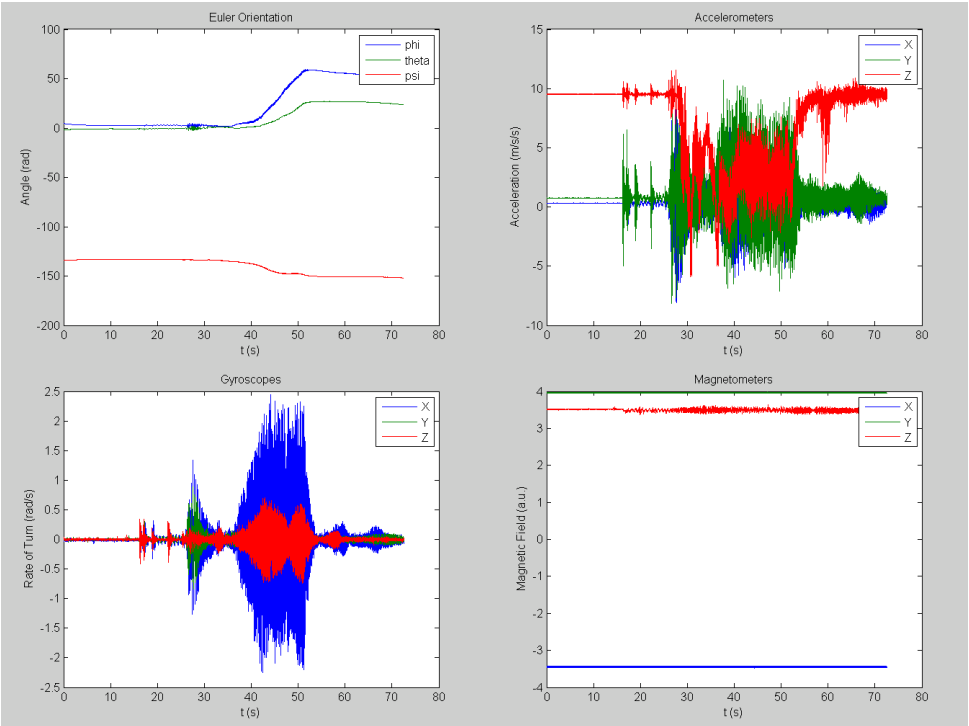


Figure 45. Adjusting the MT9-B crossover frequency. (Exp. X11).

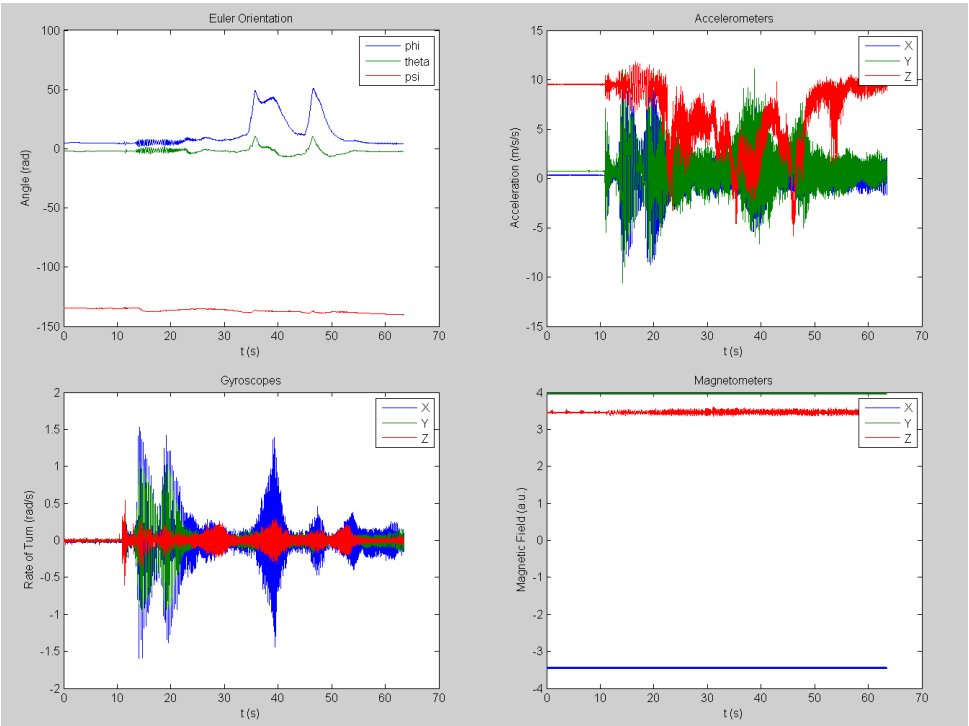


Figure 46. Magnetometer weighting set to zero. (Exp. X9).

Furthermore, as can be seen from the magnetometer plots presented later throughout this section, the magnetometer readings stay nearly constant across the entire flight duration with only a small amount of noise. Most small deviations probably represent actual heading changes while hovering the helicopter.

The magnetometer data could even be used solely to handle orientation estimation. However, this method would not be very accurate (particularly indoors). Yet, in reality, the goal is to simply hover the helicopter in place. Therefore, giving the magnetometer data extra weight could stabilize the filter's overall output and lead to better performance.

Therefore, since it seemed the magnetometers were performing well, the magnetometer weighting was set to 5.0 for a powered flight test. The results can be seen in Figure 47. Here, the orientation estimate seemed much more resilient to drift – at first. Unfortunately, eventually the orientation estimate still drifted incorrectly to one side. However, when the helicopter touched-down, the sensor corrected itself.

These magnetometer tests did not correct the problem, but did give confidence that EMI from the motor was not affecting the normal sensor operation. Increasing the magnetometer weighting would have a severely adverse affect on the orientation estimate, otherwise. Ruling out EMI corruption was one further step in correcting the orientation estimation problems.

The next step is to investigate IMU mounting and vibration isolation. The original mounting location on the rear swashplate servo provides several natural advantages. First, the mounting surface is perfectly level. Secondly, the surface is perfectly aligned with the helicopter body frame. Thus, mounting the IMU in-line with the helicopter's frame is exceptionally easy. Finally, the location is close to the helicopter natural center of gravity. Unfortunately, this location failed to produce stable results.

A decision was made to remount the IMU to the tail boom. This new location is farther away from the constantly-moving swashplate, servos, and main rotor shaft. However, the IMU is also farther away from the helicopter's center of gravity. The IMU was mounted in a simple fashion. First, a double layer of sticky foam-tape applied to the mounting location. Next, the IMU is mounted on the tape, and a cable-tie is latched around the IMU and tail-boom.

Thus, the IMU mounting has a few natural disadvantages. The mounting surface is not level and prone to IMU wobble. The IMU can be mounted along the lateral axis with some degree of accuracy, but not the longitudinal axis. Finally, the location can be jarred with significant vibration noise if the tail boom wing guard bounces.

A flight test was performed with the new mounting location. Unfortunately, the IMU drift did not significantly improve. These results are depicted in Figure 48.

Nevertheless, the tail boom still theoretically seemed to be the preferred position for mounting. Many other resarch projects have used the tail boom as the mounting location successfully. Thus, the vibration noise must be causing more problems than originally estimated.

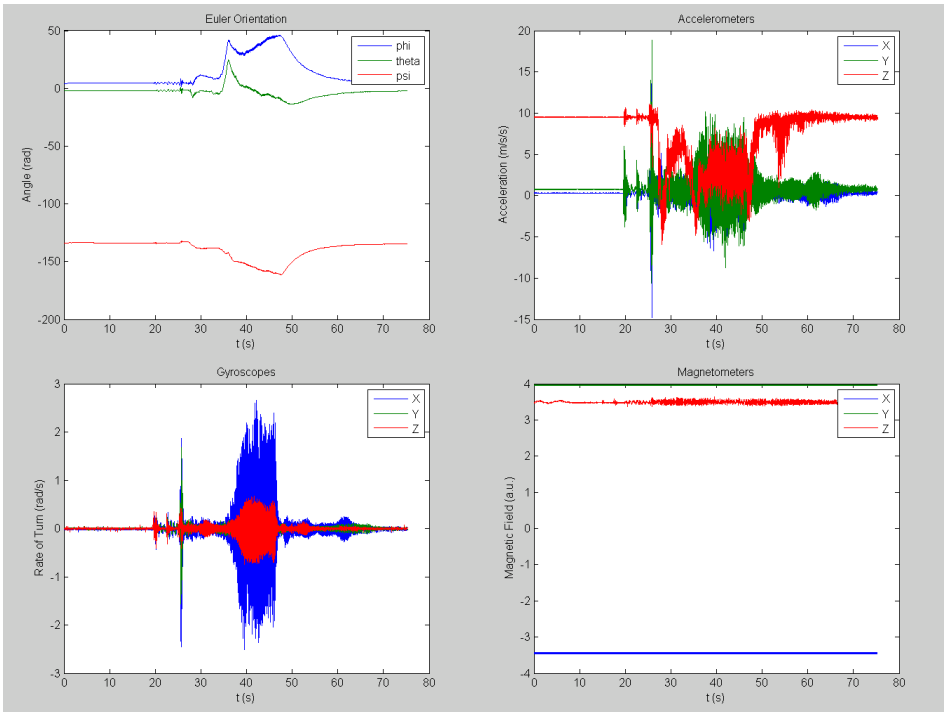


Figure 47. Magnetometer weighting set to 5.0. (Exp. X12).

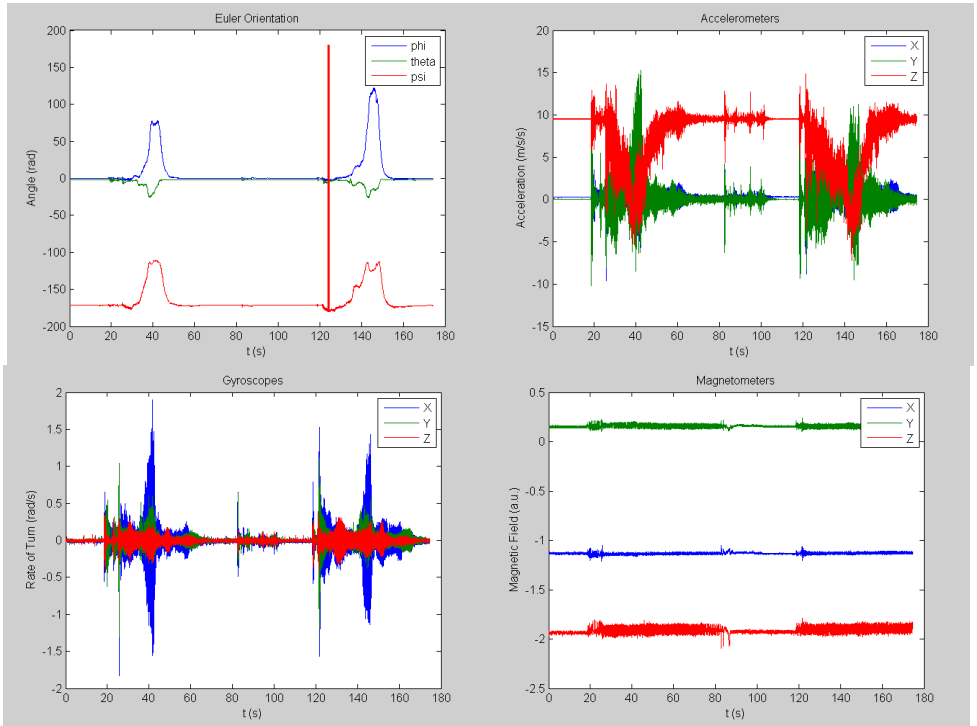


Figure 48. MT9-B mounted on the tail boom. (Exp. X14).

Finally, the assumption that the vibration padding isolation was sufficient was reevaluated. Regardless of software tuning and mounting location, the IMU was providing unreliable data. Therefore, large blocks of soft, open-cell foam replaced the foam tape attaching the MT9-B to the helicopter. This foam was sandwiched around the sensor. The primary fear with this setup is mounting error (i.e., IMU orientation offset from the helicopter frame) and over-isolation of the IMU itself – potentially leading to lagged orientation changes from the IMU.

The first experiment with the new padding also moved the IMU to the training gear cage (originally built to hold a laptop PC). The results can be seen in Figure 49. While this setup definitely provided robust results, it was quickly realized that the orientation of the training gear might not always be close enough to that of the helicopter body to provide a meaningful overall orientation estimate. In other words, the IMU was found to be over-isolated from the helicopter orientation. The training gear orientation typically changes very slowly over the course of a flight test, providing little insight into the actual helicopter orientation. Thus, the IMU was returned to the helicopter tail-boom and sandwiched in foam, again.

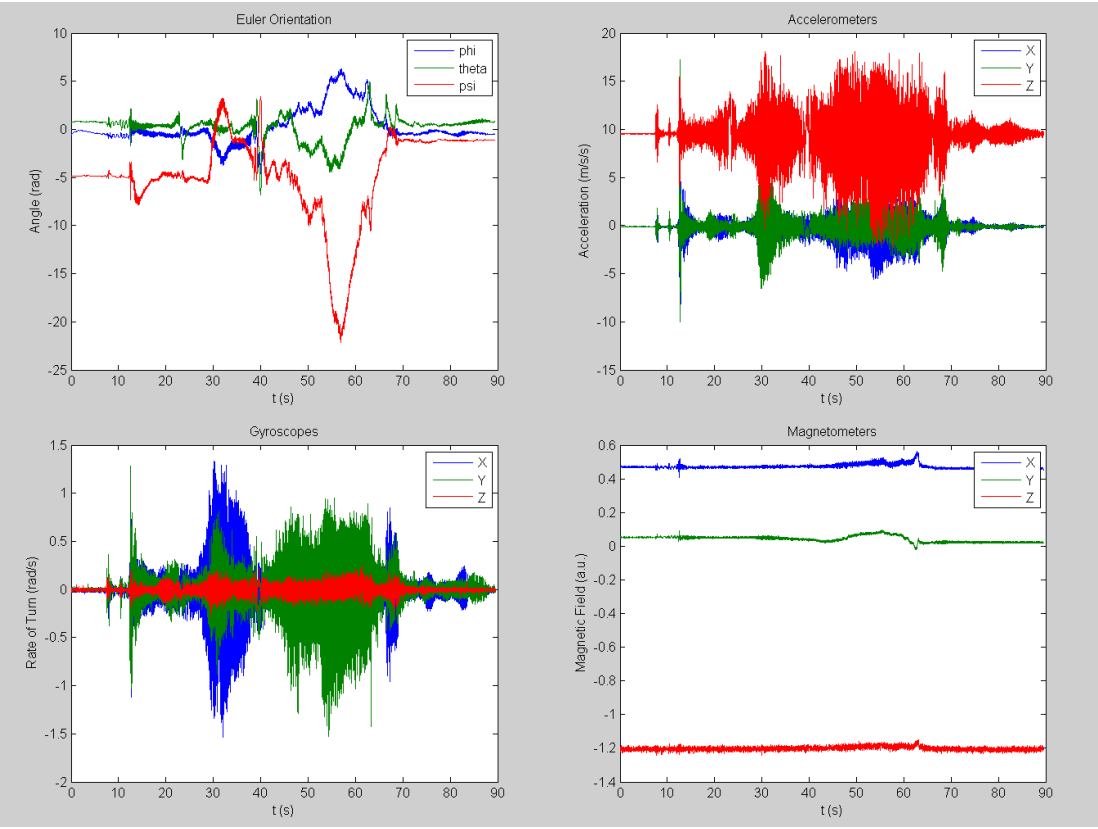


Figure 49. MT9-B mounted on the training gear with foam padding. (Exp. X17).

Results for the powered flight test with the IMU padded on the tail-boom can be seen in Figure 50. Here, after the large spike always present during take-off, the IMU settles down into normal operation. The orientation differs by only ±10° from zero – a large improvement from previous results. Since the actual orientation of the helicopter while manually flown stays within this same tight range around zero, the estimate seems to be accurate.

This result signifies a significant improvement in IMU orientation estimate. In fact, the results are good enough to warrant further experimentation into automatic flight testing. While the accelerometer and gyroscope data remains noisy much of the time, the IMU itself seems to be filtering a reasonably accurate orientation estimate from the data. The remounting on the tail-boom increased this noise level, but also allowed for much better estimation of the actual helicopter frame orientation. At this point, it is unknown whether the orientation estimate will provide sufficient accuracy to maintain an automatic hover with the helicopter control system. However, the data itself is much more promising with this latest foam-padded remount of the IMU.

Further work focused on flight-testing the helicopter using the implemented software control system.

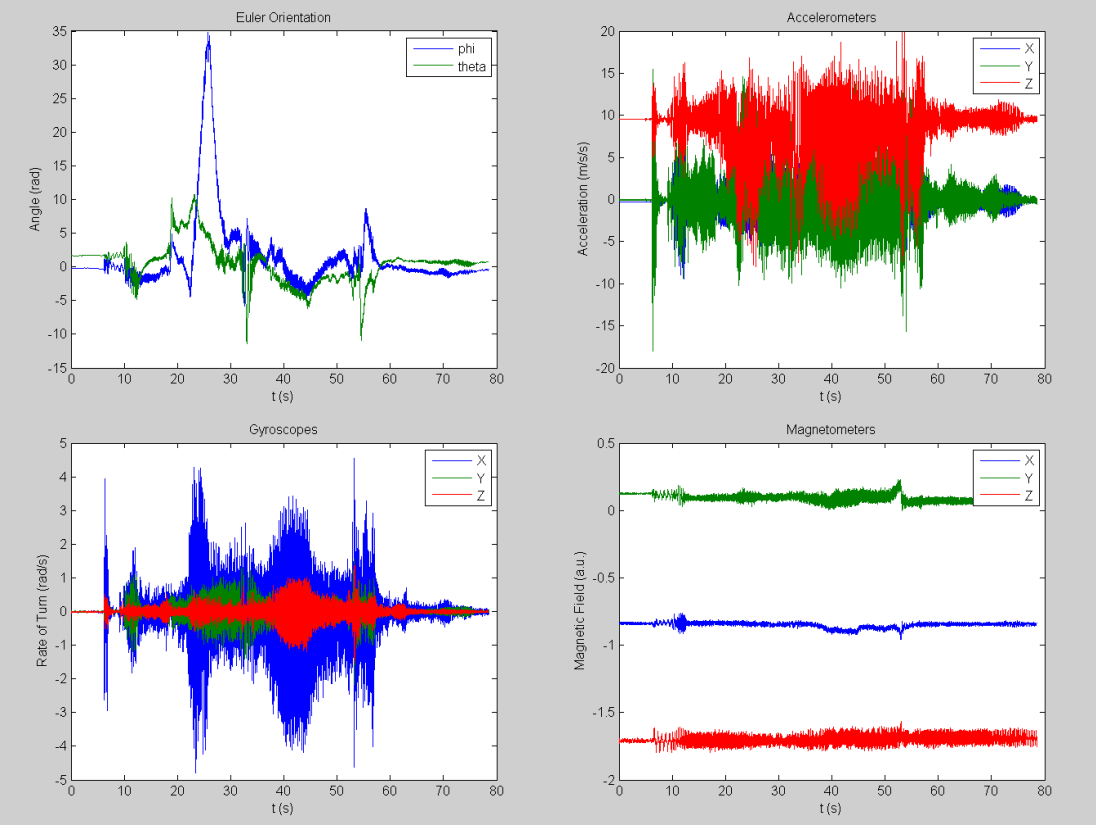


Figure 50. MT9-B mounted on the tail-boom with foam padding. (Exp. X19).

4.2.3 Flight Testing

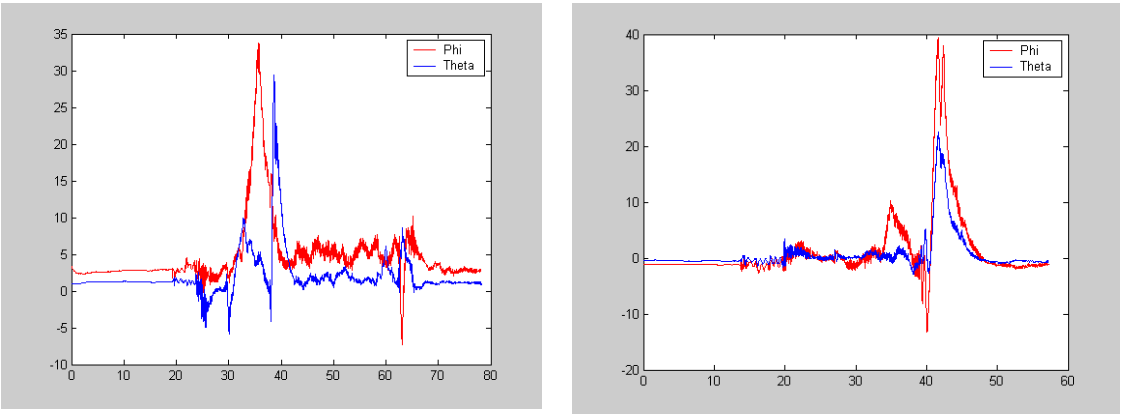
With significantly improved data from the IMU, the real flight-testing could resume. A few static tilt-test experiments were performed to test out the hardware and data-logging software. After showing promising results (i.e., actuating the helicopter swash-plate in a compensating direction relative to the tilt), flight testing began.

The first flight test with the foam-padded IMU performed very well. The pilot manually started the helicopter and brought the rotors to speed on the ground. Once the helicopter stabilized, the pilot raised it into the air in a hover. When the pilot relinquished control to the PC control system, the helicopter remained hovering in place. The helicopter was allowed to hover automatically for a few seconds before the pilot regained control and manually landed the helicopter. However, this experiment was not repeatable under the same conditions and using the same process. These experiments are depicted in Figure 51.

Because the control system did not readily repeat an automatic hovering success, troubleshooting and tuning of the control system began. These experiments are contained within the remainder of this section.

Further tilt-testing followed to troubleshoot the control system. During these tests, the PC was allowed full control over the helicopter servos. However, under automatic flight tests, the PC only controls the aileron and elevator. The human pilot always controls the throttle and tail during flights for safety. These full-control tests showed acceptable results to manual tilting of the helicopter by actuating the swash-plate in the compensating (opposite) direction.

A flight test was next performed with the current software build. Unfortunately, this test failed to produce acceptable hovering results. The PC control system continued to pull hard to the side when given control.



(a) (b)

Figure 51. Early flight tests: (a) success and (b) failed follow-up. (Exp. 4/7).

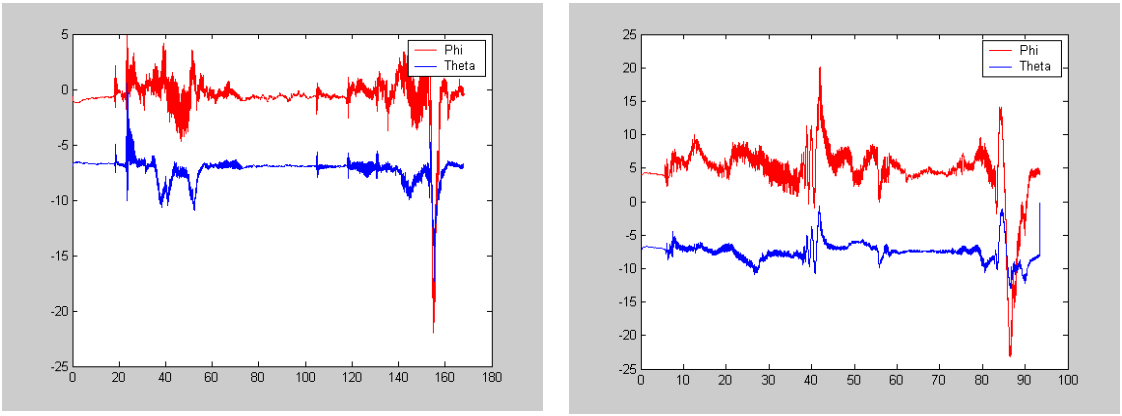
Another tilt test was performed to re-evaluate the control system performance. However, the results remained identical to the previous two tilt tests using this system build (positive). Poor IMU data must be causing the flight tests to fail.

At this point, the IMU was re-mounted with additional foam completely surrounding it and the tail boom – promising maximum isolation from vibration noise. A manual flight test was then performed to evaluate the IMU data quality. Here, the results were very promising towards future automatic flight testing. The pitch and yaw data varied very little over the course of the manual test. This experiment is depicted in Figure 52 (a).

The next step was thus a return to automatic flight testing. Here, and with the previous tilt test, the PID gains were set as follows: K_i = 2.0, K_p = 4.0, and K_d = 1.0. Furthermore, the PID was allowed to use ±40% of maximum servo throw. The limit on servo throw allows for higher PID gains and quicker response without a chance of binding the servos. Furthermore, it's unlikely a stable hover will ever require a correction of more than 40% in either direction.

This automatic flight test failed to maintain a stable hover when given control. Again, the helicopter immediately pulled hard to the side. The results are shown in Figure 52 (b).

At this point, barring recurring bad data from the IMU, it was thought that the software implementation must have a bug. Another tilt test followed to again check for correct servo directions (the IMU had been remounted in a reversed position). Next, the source code was changed to also log PID data along with orientation. A manual flight test followed to examine the generated PID outputs relative to the actual helicopter orientation during a normal hover. Given the level of noise present in the IMU data, the PID outputs seemed reasonable.



(a) (b)

Figure 52. (a) Evaluating pitch and yaw data (Exp. 9) and (b) Adjusting the PID gians to 4-1-1 did not improve flight results (Exp. 10).

A problem with the swash-plate actuation was noticed during further tilt-testing. Namely, the swash-plate was not using its full range, even when given min/max commands. Furthermore, the swash-plate behavior seemed to worsen as its collective-pitch increased (determined by the throttle stick position). Therefore, the source code was examined in an attempt to troubleshoot this problem.

Because the PCTx is a one-way system (i.e., the PC can transmit controls to the transmitter, but the PC does not know what commands the transmitter is sending to the receiver), the software "throttle" position must be set to a default value. While the actual throttle control is always maintained by the human pilot, the throttle position sent by the PC is used for the swash-plate collective-pitch mixing. The code originally set the default throttle position at zero – a potential cause of the strange swash-plate behavior under PC control. Thus, the default throttle position was increased to fifty-percent, or neutral.

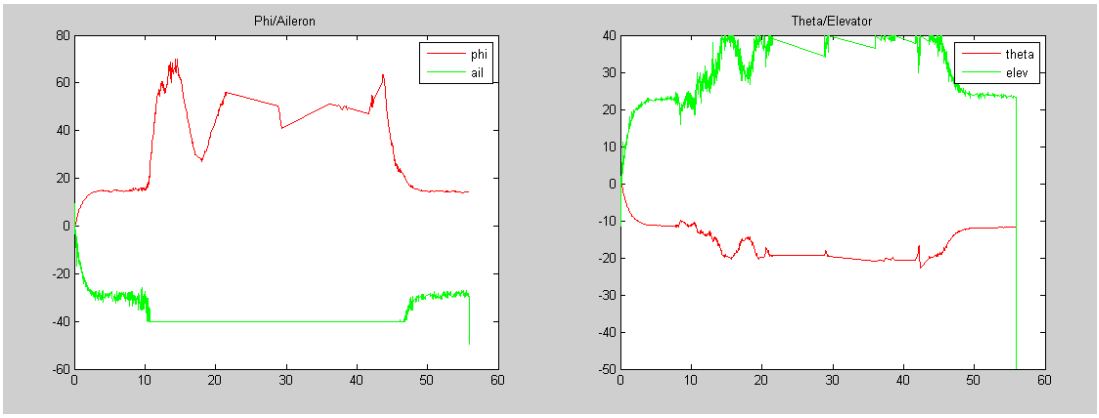
A recorded tilt test showed improved swash-plate actuation while using the increased default software throttle position. Unfortunately, the swash-plate still did not seem perfectly-actuated in response to the manual tilting. However, the decision to continue with an automatic flight test was made. Flight tests provide a better opportunity to observe the actual orientation changes of the helicopter in response to swash-plate changes, rather than merely observing the swash-plate itself during a static test.

Thus, a semi-successful automatic flight was made using the updated throttle code. The flight data is shown in Figure 53 (a). Once the helicopter was manually flown into a stable hover, the PC control system was allowed to take over for a few seconds. The flight behavior was similar to the first successful flight test. However, post-examination of the flight logs show a severe amount of IMU drift within the orientation data. In fact, the control system was providing max compensation during much of the flight. The flight was successful with poor data.

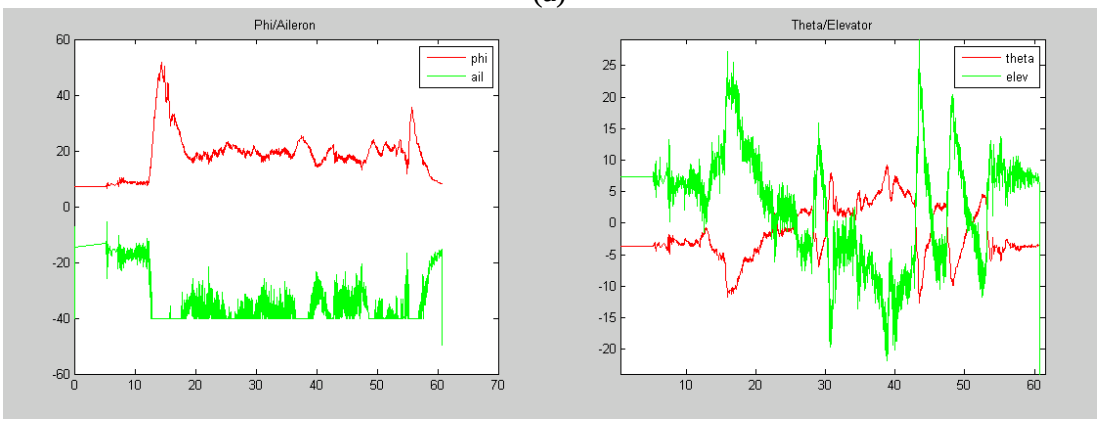
Unfortunately, the two further flight tests failed to produce similar positive results. In both of these flight tests, the helicopter immediately pulled backwards when given PC control. In the first test, the data shows significant IMU roll drift around +20°. This amount is less than the previous experiment, but more than should be present in a level hover. The pitch readings varied between –5° and +10° -- not unreasonable, but still more than should be present. The data from these flight tests is shown in Figure 53 (b,c).

The second repeat of the flight test provided similar results. The helicopter control system immediately pulls the helicopter backwards when given control. Here, the roll data possesses the same average around +20°, but the pitch estimate average is more realistic between -13° and -3°. The data itself should not be causing the control system error. In fact, the pitch compensation generated by the control system called for a forward pitch, not backward. It seemed there was a problem somewhere in the control system implementation causing incorrect swash-plate actuation.

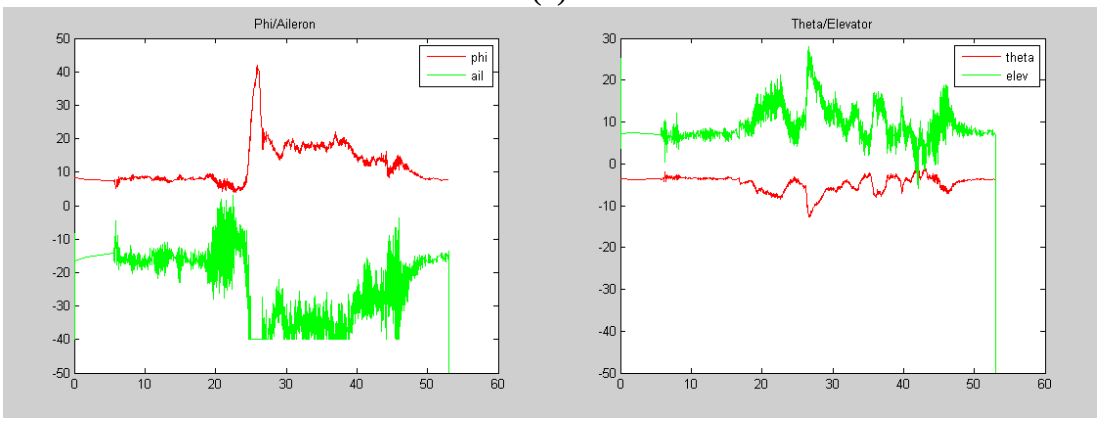
Furthermore, when looking at the orientation data, the roll average was farther from zero during a hover than expected. Thus, an examination of the collected orientation data was made in the hopes of improving the control system PID setpoints.



(a)



(b)



(c)

Figure 53. (a) Semi-successful flight test over short duration; (b) follow-up tilted backwards when given control; and (c) continued tilting backwards when given control. (Exps. 15-17).

The next phase of testing focused on remounting the IMU yet again and also better adjusting the setpoints. The recurring behavior of pulling hard backwards during automatic flight could be chalked up to poorly-specified setpoints. Because the IMU is mounted on a circular tube, its orientation is not perfectly level relative to the helicopter frame. Furthermore, the setpoint specification of zero for perfectly neutral might not be the optimal position in a stable hover.

The IMU was returned to its original position atop the rear-most swash-plate servo (directly behind the swash-plate itself). This provided a precisely level mounting surface relative to the helicopter frame. Furthermore, previously recorded flight data was parsed and analyzed for optimal setpoints. A table of collected data and some statistical results is shown in Table 6. At this point in experimentation, the data indicated a setpoint of 6.6 for roll and 0.4 for pitch.

Unfortunately, these changes did not fix the immediate lurch backwards when switching to automatic control. Two flight tests were performed using this new setup that did not yield improved results (shown in Figure 54). Both of these experiments showed the same problem when relinquishing control to the automatic control system: a large spike in the orientation as the helicopter lurched. Unfortunately, very little information can be gleaned from the data logs resulting from these two experiments. The control system's severe reaction when given control indicates a fundamental problem within the implementation. The odd behavior of the swash-plate during tilt testing was recalled and pursued further as a potential culprit.

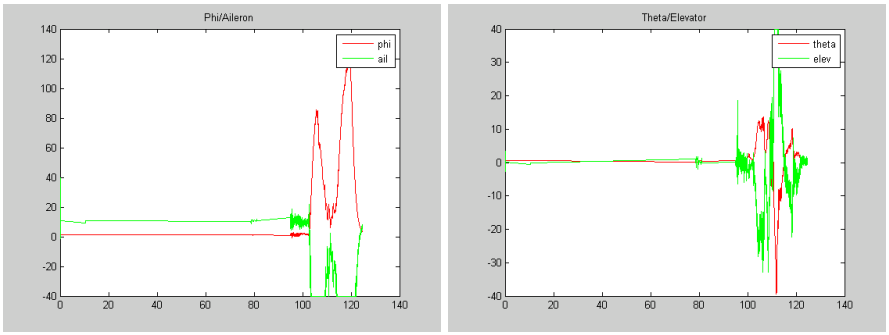
A suspicion began to arise that the transmitter was not generating correct servo commands given PC inputs. While all previous tilt tests produced seemingly good results, the flight tests invariably failed. The difference between these two situations was the amount of control given to the PC. In the tilt tests, the PC was typically allowed full control over the helicopter servos (including throttle, and tail). Full control was given to test the collective-pitch mixing quality calculated by the PC. However, when performing actual flight tests, the human pilot always maintained control over the throttle and tail servo for safety and smooth transitions between manual/automatic control. To achieve limited control, the R/C transmitter was setup in Trainer mode to allow MASTER control of THR/RUDD and SLAVE control of AIL/ELE.

However, upon close examination of the swash-plate actuation under full and limited control, the results were invariably different – even though the control inputs generated by the PC were identical! The only course of action was to examine the exact servo outputs generated by the transmitter given PC inputs. Then, new formulas could be derived to produce correct swash-plate action under limited control.

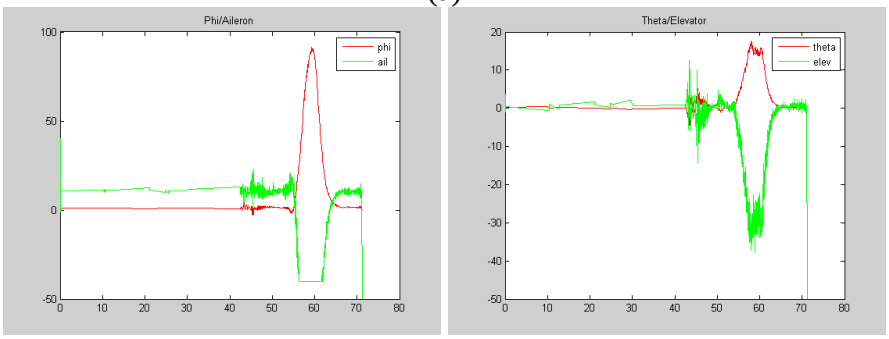
This new data analysis task was rather tedious to say the least. Luckily, the answer was found early with a simple realization, but without examining the exact PPM outputs generated by the transmitter. The transmitter has a built-in monitoring display of current servo positions. The servos are presented in a simple gauge format with neutral at center and tick marks at 25% intervals. Given a valid PPM frame from the PC (i.e., servo positions properly mixed), the servo monitor should exactly match the servo inputs. Under full control, the inputs/outputs matched precisely. However, under limited control the servo monitor showed completely different servo positions in many cases!

Table 6. Finding Optimal Setpoints

FLIGHT		Phi	0.5		Theta	0.5
#	Mean	Var	ThrMean	Mean	Var	ThrMean
xsens40	6.5749	1.2028		0.1999	1.3896	
xsens39	6.6242	1.8615		0.66	1.2404	
23	12.4105	388.66	11.6134	0.6534	8.5364	0.6441
24	10.6431	102.6343	9.9831	0.6861	5.0055	0.6591
25	13.3601	248.6122	12.7071	0.817	8.0782	0.783
26	10.2563	273.1216	9.5524	-0.0921	9.5923	-0.0701
27	10.273	52.9112	10.1677	0.5981	3.7649	0.598
28	14.3923	130.182	14.4634	-0.2282	4.4367	-0.2419
29	12.9364	36.1044	13.0866	1.0786	5.4904	1.057
30	17.3928	41.0228	17.6787	2.6297	4.5844	2.6738
31	19.0567	75.048	19.627	1.5154	18.2588	1.4492
32	10.7165	115.6384	10.9082	2.5885	9.3809	2.5816
33	10.6547	37.2822	10.7221	1.8708	5.8369	1.8701
34	16.6841	122.1712	17.0857	0.5817	8.3857	0.5741
	12.28397		13.13295	0.968493		1.048167



(a)



(b)

Figure 54. (a) Unsuccessful automatic flight test with IMU moved to rear servo and setpoints adjusted; and (b) repeat with remounted IMU and adjusted setpoints. (Exps. 18-19).

While undocumented in the user manuals, the transmitter assumes different input schemes between limited and full SLAVE control. It is unknown whether this behavior is a feature or bug. In any case, under limited control the transmitter assumes the PPM signal from the SLAVE is completely unmixed and untrimmed. Therefore, while nine channels of PPM data are sent, the transmitter only uses four to control the helicopter: throttle, aileron, elevator, and rudder (default channels 1-4). The other channels are ignored.

In fact, the transmitter essentially behaves as if the PPM channels are raw stick inputs rather than servo data. Thus, half-stick (neutral) is represented as 1500-ms, min-stick as 1000-ms, and max-stick as 2000-ms. While this scheme greatly simplifies the PC control system calculations by removing the necessity for software servo mixing, it also provided a severe troubleshooting headache.

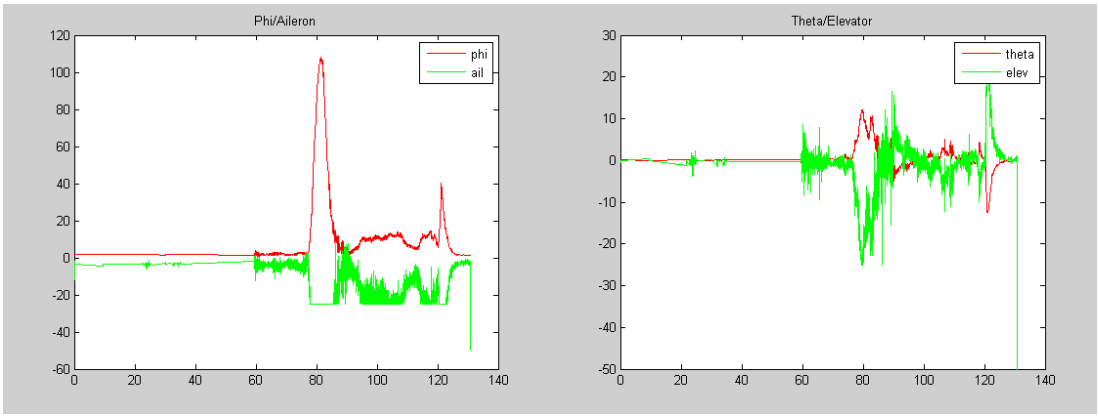
After removing the servo mixing from the source code and sending simple aileron/elevator commands instead, the system was ready for more testing. Three tilt tests followed to verify correct directions and swash-plate actuation. Additionally, the maximum throw of the swash-plate servos was reduced to ±25%. The removal of the erroneous servo mixing code allowed for full-range motion of the swash-plate again, requiring a tighter limit on maximum throw. Finally, the setpoints were also returned to zero.

The time had come to return to automatic flight testing. Three troublesome flights began this round of testing. These experiments are depicted in Figure 55. However, based on the logged orientation data for the latter two tests, it seemed to be partly the unreliable IMU producing poor data under high-vibration again.

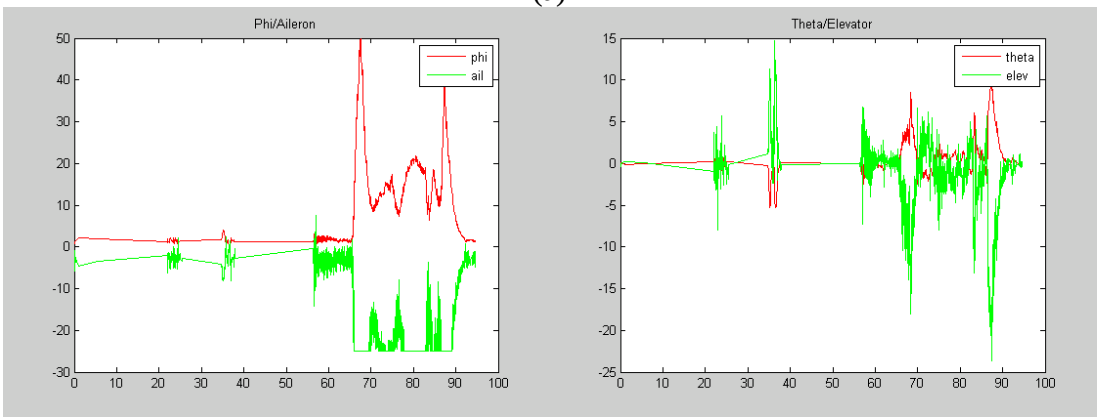
In the first flight test, the orientation data seemed reasonable given previous flight data. The roll averaged around +10° while pitch averaged around 0°. However, the flight did not automatically hover successfully. Post-examination of the flight logs shows far too much roll compensation given the rather small +10° error. In fact, the control system pushed the roll to the safety limit much of the flight. Here, the problem can most likely be attributed to poor PID loop tuning and the removal of the calculated setpoints. A setpoint of zero for roll seems to be incorrect in a stable hover.

The next two flight tests both showed much spikier data than the first test. Several factors were more than likely contributing to the spiky data. First, the PID control system is obviously over-compensating and keeping the roll/pitch at the limit erroneously. Secondly, the helicopter pilot might be having more difficulty putting the helicopter in a stable hover, here. With the helicopter not in an optimal position before the control system is given control, the chances are less likely that it will maintain the hover. Finally, the IMU could be generating worse data than before. Sometimes, the IMU will perform very poorly when the manual take-off is rougher than usual. Furthermore, the IMU will take much longer (if ever) to settle into normal operation.

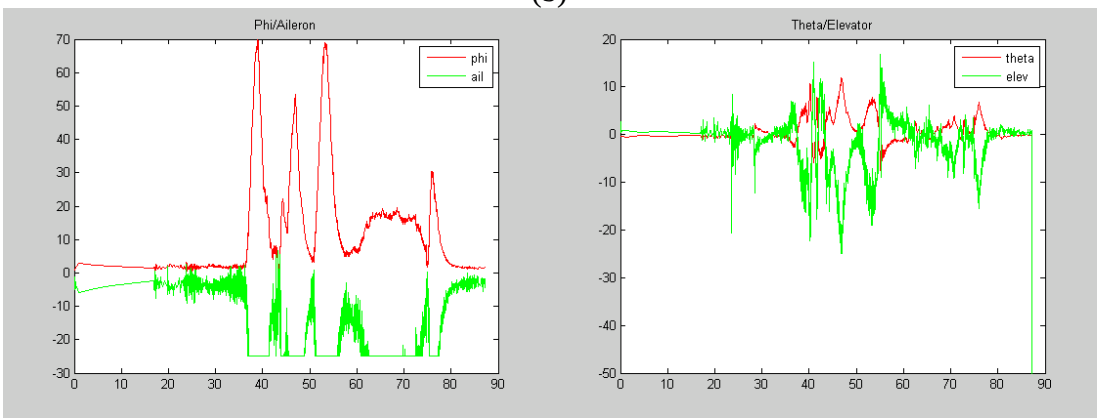
Given the roll average around +10°, the new setpoints of zero were obviously not going to be successful. Correcting the setpoints is the first step towards improving the control system performance. Next, the PID tuning and limits needs to be evaluated towards better operation. While fast response is desirable, constant over-compensation can cause disastrous results with a vehicle such as a helicopter.



(a)



(b)

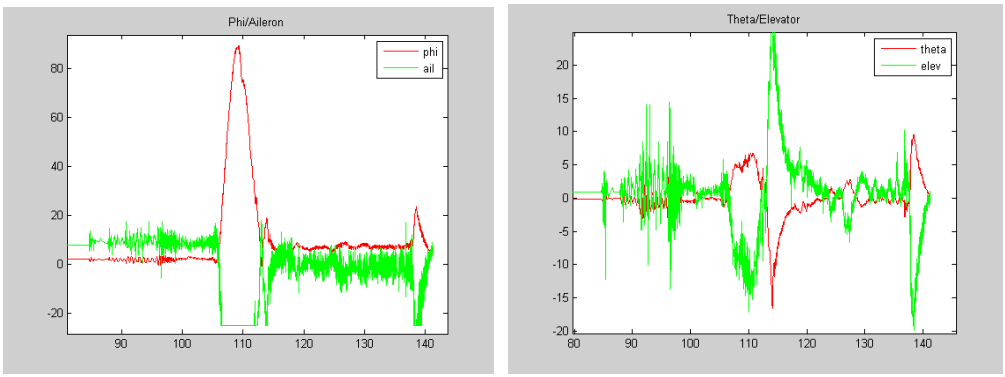


(c)

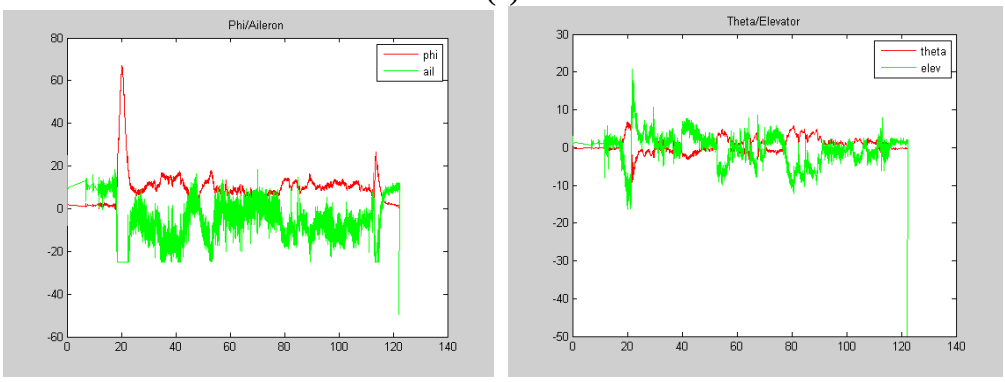
Figure 55. (a) First flight test after removing software servo mixing; (b) repeat of flight test indicating possible bad data from IMU; and (c) third test flight after removing servo mixing. (Exps. 23-25).

Returning to the calculated "good" setpoints improved the next two automatic flight tests. These tests are shown in Figure 56. The first flight test provided adequate hovering results when given control, while the second test failed to maintain the hover after a few seconds. Both tests show a roll/pitch averages very near the setpoints and much more stable orientation data. However, the second test does appear to possess slightly noisier data, particularly when comparing the PID outputs between the two flight tests. Also, the huge spike during take-off is very obvious here.

While closing in on repeatable automatic flights, the system was far from reliable. At this point, the return to a focus on optimal PID parameters was possible. The implemented PID loop uses only the estimated orientation data supplied by the IMU. While it is possible to include gyroscope data (change in orientation), the orientation was deemed much more reliable and less noisy. With this arrangement, the setpoints less the orientation represent the integral error terms. The proportional term represents the change in orientation (current orientation less previous orientation). Finally, the derivative term represents the second-derivative of orientation. Of these three terms, the current orientation is most important for a stable hover.



(a)



(b)

Figure 56. (a) Successful flight test with adjusted setpoints to statistically-determined values; and (b) repeat of flight test with degraded performance. (Exps. 26-27).

For the next flight test, all gain parameters were toned down. The system was simply behaving too erratically with high gains and the corrected servo mixing. The gains were adjusted as follows: K_i = 0.5, K_p = 1.0, and K_d = 0.25. The maximum throw remained at ±25%. This gain configuration puts the most weight on the proportional gain term, or the rate of change in orientation. Since maintaining the orientation is the top priority, and orientation errors worsen exponentially over time, leveling the orientation as quickly as possible is imperative to a successful flight test. The other two terms were lowered proportionally.

The automatic flight test seemed to succeed using these new values, but improvement was still needed. Here, the pitch behaves very well – varying merely between -4° and 6° over the course of the flight. The roll also remained steady throughout the flight (after the initial take-off spike), but average closer to +15°. The roll average is more than likely too far from the designated setpoint to provide adequate and reliable automatic hovering here. However, this behavior is probably the fault of the IMU not settling down after take-off. The results of this experiment are shown in Figure 57.

The next test adjusted the PID gain parameters as follows: K_i = 1.0, K_p = 0.5, and K_d = 0.25. The increased gain on the most stable and relevant parameter (integral error or simply orientation) seemed a logical choice. Here, the goal is to correct orientation errors quickly through the integral gain term or reset. This method has the advantage of working on real orientation data instead of estimated derivative data from multiple data points. The hope is to rigorously hold the orientation at the setpoints.

However, the automatic flight test did not perform as well as hoped. Yet, the test still managed to semi-successfully hold the helicopter in a stable hover when control was relinquished. Post-examination of the data showed similar results to the previous flight test. However, the pitch varied a bit more erratically than before – spiking as much as –10° and +10°. The roll remains a good amount above zero as in previous experiments. The results of this experiment are shown in Figure 58.

A further flight test followed using this software build without repeated success. Based on the quality of the data, the IMU, and the helicopter flight behavior, the setpoints still seemed like they could use some adjustment. The results of this experiment are not included here for redundancy. The table of mean/variance of the flight data was updated and re-examined to find better setpoints.

Next, the roll setpoint was adjusted to 12.0 and the pitch to 1.0. These new setpoints were based on a much larger dataset comprised of fourteen different flights. However, the presence of heavy-noise periods and completely static periods (i.e., before taking-off and after landing) must have skewed the data somewhat. The next flight test was not successful using these new values (Figure 59).

These three flight tests led to some initial conclusions. First, the lowered PID gains improved the flight results but did not instantaneously stabilize the flight performance. Good setpoints must exist for the helicopter in a stable hover, but these setpoints seem to depend much on the initial hovering position reached by the pilot and the amount of noise accrued by the IMU during take-off. Further experimentation will examine the PID parameters in detail.

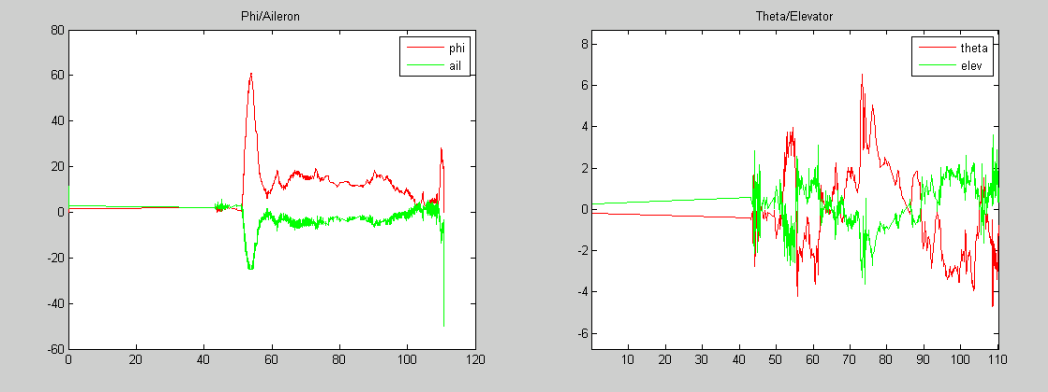


Figure 57. Flight test with adjusted PID gains: success (Exp. 28).

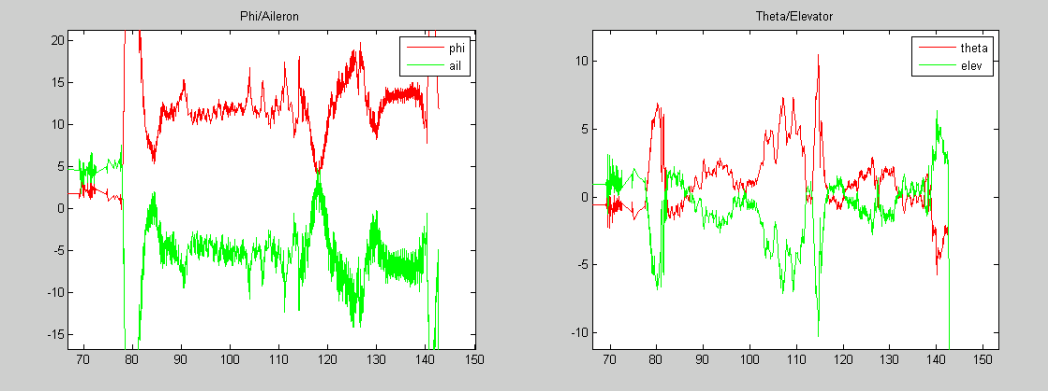


Figure 58. Semi-successful flight test after adjusting PID gains (Exp. 29).

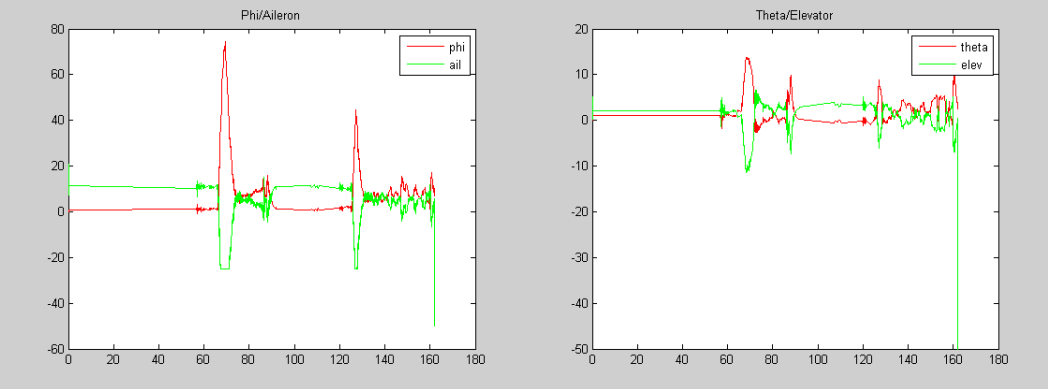


Figure 59. Adjusted setpoints to compensate for IMU drift. (Exp. 32).

The new setpoints were not consistently improving flight results. Therefore, the original "good" setpoints were returned to use. These setpoints were determined from only two manual flight tests, but both tests were very stable hovers.

Returning the setpoints to the previous 6.6 roll and 0.43 pitch led to a semi-successful flight test. A repeat of the flight test was also semi-successful. In both of these flight tests, the helicopter hovered for the duration of the automatic flight. However, the hovering quality degraded over time, and the pilot had to renew manual control. The results of these flight tests are shown in Figure 60.

As can be seen from the data logs, the hovering was stable for the majority of the flight (after take-off spikes). Furthermore, observing this same data shows the helicopter holding a position not exactly at the setpoints. This behavior is most obvious in the roll terms. While the setpoint is 6.6, the actual flight positions seems to waver around 10.0 (or higher in the repeat flight test).

While the flight tests were becoming increasingly successful, the search for optimal PID parameters was still underway. The setpoints still seemed a bit off for a stable hovering position. Thus, after simple inspection of the data log files, the setpoints were adjusted to 10.0 roll and 1.0 pitch. These numbers were taken from simple visual estimation of the data plots over time. The remaining PID parameters were left the same to test these setpoints alone.

Again, the automatic flight test was successful, but with degrading performance over time. The results of this flight test can be seen in Figure 61 (a). These setpoint changes were making subtle modifications to the flight behavior, but the overall results were not approaching stability. Therefore, the gain parameters were re-examined and adjusted again to help cope with the amount of noise present in the system during flight testing.

The theoretical goal of automatic hovering is maintaining a level flight (zero integral error) while holding the helicopter "still" (zero angular velocity). Thus, reacting quickly to compensate for angular velocity (proportional term) is an important part of automatic hovering. Thus, the proportional gain was increased again to attempt to stabilize the hovering consistently.

In the next experiment, the PID parameters were set to, $K_i = 0.25$, $K_p = 1.0$, and $K_d = 0.25$. This experiment is essentially a repeat of previous one, but using the new human-estimated setpoints of 10.0/1.0. The hope is that less gain will be needed on orientation at any given time due to the setpoints being closer to true hovering position.

These parameter changes increase the sensitivity to angular velocity while decreasing the rate that the orientation is leveled-out. This flight test produced the same level of semi-successful but slowly-degrading results as previous tests. Flight test results can be seen in Figure 61 (b).

The initial results based on this round of experimentation show that the flight behavior is staying consistent in most cases. The parameter changes are having small impacts on the overall system performance. However, the system is hovering itself for short durations given a good starting position by the pilot. Some fundamental issues are holding back significant progress and will be tackled in the next round of experiments.

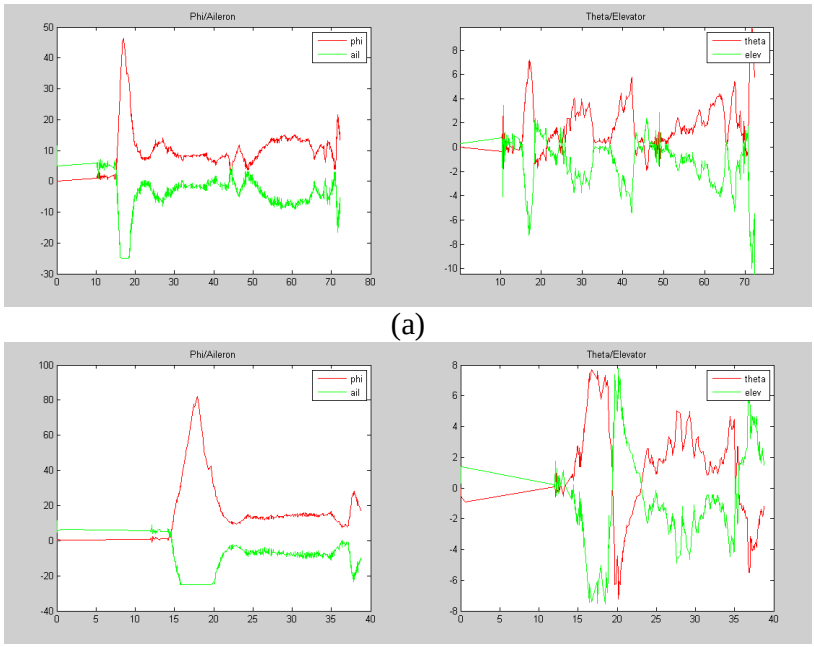


Figure 60. (a) Semi-successful flight test with returned setpoints to statistically-determined values; and (b) successful repeat of flight test. (Exps. 33-34).

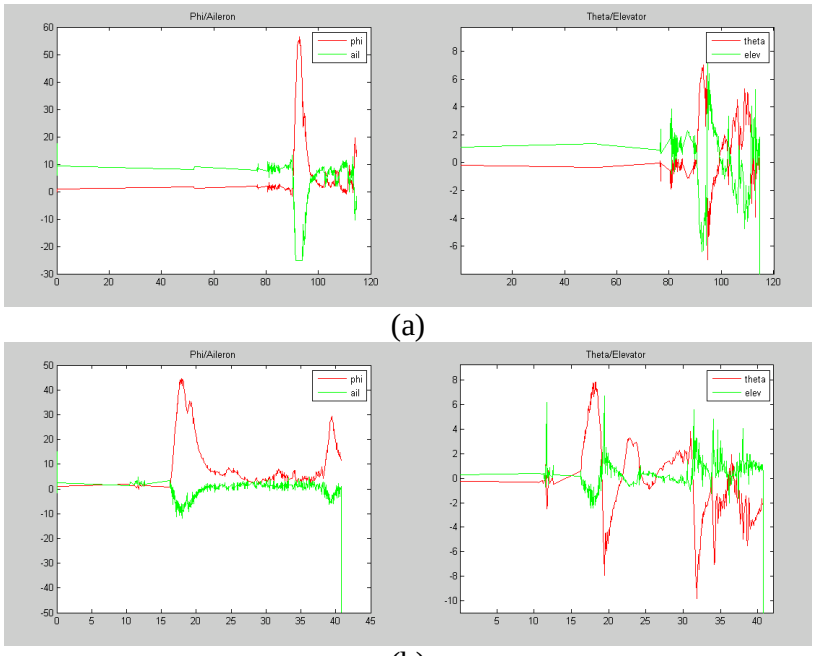


Figure 61. (a) Setpoints gains adjusted again to 6.6/0.43: no flight improvement seen (Exp. 35), and (b) PID gains adjusted to increase P term: results degrade over time (Exp. 36).

Adjusting the PID parameters was not making enough impact on improving the flight quality. Furthermore, with the system bogged down with the weight of the safety gear and tethering, flight performance is severely handicapped. The helicopter flying in open-air without safety gear is nimble and fast. However, the weight of the safety gear not only requires full throttle just to manage lift-off, but also creates incredibly sluggish response to controls. Unfortunately, this is a necessary evil to allow for automatic flight-testing without serious risk to the helicopter or pilot.

However, some optimization was made to improve the overall flight performance. First, the source code was heavily optimized in many aspects. The servo controller interface code was simplified greatly (possible now that software servo mixing was unnecessary). All calculations for servo mixing were set to activate only when a FULL_CONTROL switch is set. The main loop now required far fewer calculations before sending the PID output to the servo controller.

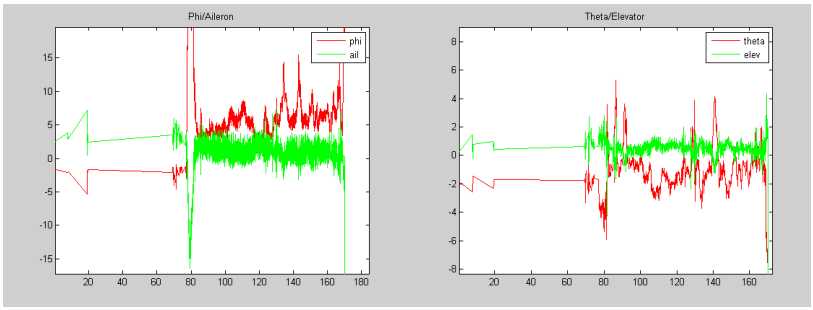
Secondly, the training gear was slimmed down quite a bit. The center mounting cage was removed completely. Not only does this remove excess weight, but also moves the long legs of the safety gear much closer to the helicopter center of gravity. The increase in responsiveness during a flight is instantly noticeable. However, there is an added risk to the helicopter's tail rotor since only a flimsy carbon-fiber wing protects it now.

These changes led to much more sensitive and responsive automatic flight tests. However, the system was also very difficult to manually control now. Simply taking-off and putting the system in a controlled manual hover was challenging for the inexperienced pilot. Three flight tests were performed using this optimized setup with moderate success. The results of these flight tests can be seen in Figure 62.

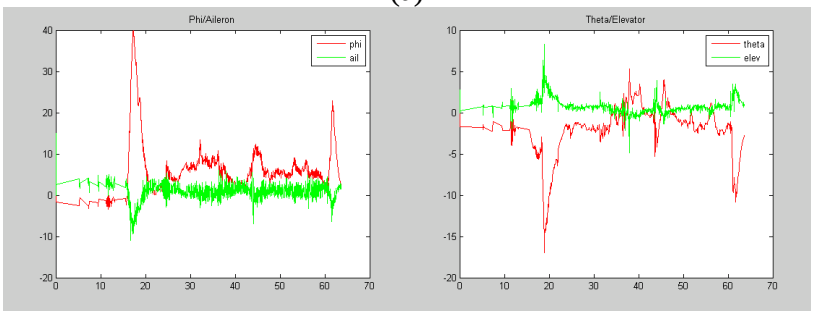
Only one final flight test was performed on the helicopter control system. To better isolate and verify the PID control quality over the helicopter, the transmitter was setup to allow only automatic aileron control. The pilot maintained control of the elevator at all times. This arrangement provided a much better environment for examining the control system response. While the pilot kept the pitch level, the control system maintained the roll successfully. The results of this flight test can be seen in Figure 63.

A final summary table was created using each logged data set. The mean orientation values and variance over the duration of the flight are recorded in the table. Mean values close to the setpoints (or zero) with low variance generally denote more successful flight tests. However, some deviations in the data might be the result of pilot error during manual flight. The flight data is summarized in Table 7.

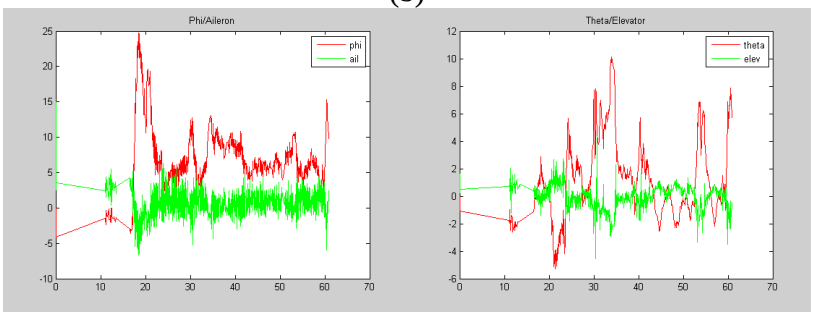
From these experiments, the conclusion can only be that the system can work successfully under ideal conditions. Unfortunately, the second conclusion is that the chosen commercial-off-the-shelf inertial measurement unit (Xsens MT9-B) is simply inadequate for the high-frequency, high-speed application of a helicopter control system. The orientation data suffers from severe estimation errors that can lead to unreliable flight performance. However, when the system the system can hover itself given a "good" starting position by the human pilot.



(a)



(b)



(c)

Figure 62. (a) Flight test after source code optimized and training gear removed; (b) repeat, manual control is very difficult with the removal of the training gear lower tier; and (c) another repeat of the flight test without significant improvement. (Exps. 37-39).

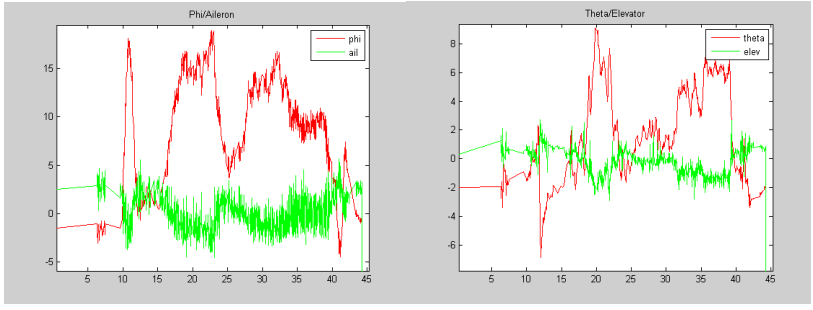


Figure 63. Manual elevator control with automatic aileron control: successful. (Exp. 40).

Table 7. Flight Test Data Summary

Log	**Endpoints**		**Phi**		**Theta**	
#	*Take-Off*	*Landing*	*Mean*	*Var*	*Mean*	*Var*
4	20		5.6841	28.4877	2.4978	14.7797
7	15		2.289	52.5553	1.1433	14.7528
9	20		-0.613	5.0026	-7.3556	1.3108
10	5		4.7307	20.3411	-7.5548	2.2033
12	65		5.2661	2.1097	-8.1343	1.064
15	10		36.3852	299.7846	-15.6766	9.1812
16	20	55	18.8856	4.4358	0.3482	16.4929
17	28	50	15.2465	10.0514	-4.6647	2.9428
18	105		49.1307	1437.9	-1.9184	118.4598
19	55		32.8066	1105.7	6.9946	38.2923
23	90	120	9.0791	9.238	0.2632	2.1569
24	70		12.9111	44.6673	0.9786	5.6038
25	35		16.7137	270.2846	1.0193	10.0979
26	115	138	6.8238	1.0384	-0.5622	2.743
27	22	115	10.114	10.9932	0.6004	3.4925
28	60	100	13.9602	6.219	0.2804	4.7914
29	85	140	12.3982	7.6001	1.467	4.3911
30	45	90	18.172	14.3369	2.8902	3.4523
31	35	65	20.7161	44.8039	3.0226	7.173
32	130	160	7.122	5.0088	2.163	2.3724
33	20	70	9.626	7.6372	1.536	2.1718
34	20	35	14.8508	13.6522	1.1042	7.2841
35	97	113	3.7306	5.3131	1.1783	4.1213
36	22	38	5.0645	2.7509	-1.0951	8.4
37	85	169	5.7126	4.9155	-0.8797	1.6371
38	20	60	5.7566	6.5115	-1.1133	4.9279
39	20	60	6.8948	8.2681	1.2821	8.5819
40	15	40	11.0748	12.7096	3.2342	6.8549

5. Conclusions

5.1 Summary

Chapter one introduced the need for a mobile wide-area surveillance platform towards intruder detection and tracking. A sensor/interceptor placement planning system was proposed to accomplish this task. Next, the advantages of a miniature UAV helicopter were described for this application – namely, complete 3D positioning capabilities. The introduction went on to propose a control system for an autonomous surveillance UAV helicopter.

Chapter two comprises the literature survey performed before undertaking the research objectives. This section sought to survey some of the most complete multi-robot and multi-agent control systems currently implemented, with particular focus on unmanned aerial vehicle (UAV) platforms due to their natural aptitude of wide-area surveillance. Most likely the best system surveyed, in terms of completeness, was the COMETS system, which sought to create a coordinated multi-UAV system capable of detecting forest fires. However, the design of the system was open enough to allow expansion into many other applications. Unfortunately, the consideration of ground vehicles was not included in the COMETS program.

Next, current UAV helicopter systems were surveyed and described briefly. Additionally, a survey of commercial autopilots developed specifically for miniature UAVs was included. This autopilot survey concluded with a best-purchase suggestion at the time of writing.

The sensor placement planning literature survey essentially describes the root of the research, the Art Gallery Problem, and then extends into the notion of visibility – used heavily during the sensor placement planning presented here.

Because of the desirability of commercial off-the-shelf (COTS) parts, R/C vehicles make an excellent target mobility platform. However, control interfaces are generally primitive and lack any sort of telemetry.

Chapter three describes the design of the sensor placement planning system and the UAV helicopter control system in several parts. The sensor placement planning system is comprised of Erdem and Sclaroff's Radial Sweep algorithm for visibility polygon generation, fixed/PTZ camera modeling, a basic branch-and-bound search, and a shortest-path interceptor placement based on Voronoi diagrams and Dijkstra's algorithm.

The UAV helicopter control system is based on the proportional-integral-derivative control algorithm. The PID design was proposed as the best method for controlling the helicopter swash-plate. An inner loop PID for attitude control and an outer loop PID for position control (similar to a receding horizon controller) comprise the navigation system. The next section describes the complete control system implementation using a standard notebook PC and commercial off-the-shelf (COTS) hardware. Two similar setups were tested. The first, more practical approach maintained the bulk of the hardware onboard the helicopter and used a USB-based servo controller instead of the R/C receiver. The second, more robust approach moved the PC off-board and sent all commands through the standard R/C transmitter/receiver pair. Both setups

had distinct advantages and disadvantages for both experimentation and real-world applications.

5.2 Future Work

Future work on the sensor placement planning would include optimization of the code for more complex areas and an extension to a patrolling mobile interceptor. Eventual work could also lead to 3-D modeling of areas for more realistic simulation.

The UAV helicopter-interceptor waypoint-navigation control system is not yet complete. Future work includes implementation of more robust hardware, a better sensor fusion algorithm, further integration of imaging techniques, and extension to full waypoint-navigation. Additionally, work on an adaptive, neurak-etwork-based PID controller should be continued to improve the helicopter response under different flight conditions.

5.3 Final Conclusions

A sensor placement planning system and UAV helicopter control system were proposed and presented here. The sensor placement planning system is able to optimize coverage versus total cost given an area, constraints, and a list of available sensor parameters (field of view, pan/tilt angles, cost, etc.). The system can process an area quickly using Erdem & Sclaroff's efficient Radial Sweep algorithm and breadth-first best mask searching. Additionally, an interceptor can be placed within the same area to minimize interception time to any point on the perimeter using the same visibility techniques in conjunction with Dijkstra's algorithm and the area's Voronoi diagram. . Additionally, a face-detection and localization system was developed to test the sensor placement planning results on real-world areas. Therefore, the real-world implementation of this system requires only the interceptor helicopter system to be complete.

The UAV helicopter control system is able to hover with limited success given a good starting hover position from the human pilot. Unfortunately, due to spiky orientation data during take-off, the control system is unable to automatically take-off. The final conclusion is that the chosen inertial measurement unit is critical to the design of a helicopter control system. Here, the IMU produced inconsistent results leading to non-repeatability of some experiments. While the system can hover the helicopter under ideal conditions, the hardware-sofware setup is non-optimal for a robust helicopter control system.

References

"9303 Heli Adv R649 & 4-8311 Digital Servos," JR Radios (Horizon Hobby), Retrieved June 2008. <http://jrradios.com/Products/Default.aspx?ProdID=JRP9252**>.

"Collective Pitch and CCPM," Heli Hobby, Retrieved June 2008. <http://www.helihobby.com/html/collective_pitch_and_ccpm.html>.

"Co-Pilot Flight Stabilization System," FMA Direct, Retrieved June 2008. <http://fmadirect.com/detail.htm?item=1489&section=20>.

"Die originalen brushless Außenläufermotoren," actro Die originalen brushless Außenläufermotoren!, Retrieved June 2008. <http://www.actro.de/de/allg/index.html>.

"FS8CPI – P.C. Ready," FMA Direct, Retrieved August 2007. <http://fmadirect.com/detail.htm?item=1770&section=29>.

"Futaba GY401 Gyro w/S9254 Digital Servo," Futaba (Hobbico), Retrieved June 2008. <http://www.gpdealera.com/cgi-bin/wgainf100p.pgm?I=FUTM0808>.

"future-universal: the universal controllers and speed governors for the brushless and sensorless generation of motors," schulze elektronik gmbh, Retrieved June 2008. <http://www.schulze-elektronik-gmbh.de/index_uk.htm>.

"HC12 Overview," Freescale Semiconductor, Retrieved June 2008. <http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=0162462LcRC5dj>.

"JAUS Domain Model (DM)", Joint Architecture for Unmanned Systems (JAUS), Version 3.2, Retrieved August 2005. <http://www.jauswg.org/baseline/Domain%20Model%20v3.2%2010Mar05.doc>.

"JAUS," Joint Architecture for Unmanned Systems (JAUS), Retrieved June 2008. <http://www.jauswg.org/>.

"JAUS Reference Architecture (RA)", Version 3.2, Joint Architecture for Unmanned Systems (JAUS), Retrieved August 2005. <http://www.jauswg.org/baseline/refarch.html>.

"JAUS Tutorial Presentation", Joint Architecture for Unmanned Systems (JAUS), Retrieved June 2008. <http://www.jauswg.org/JAUStutorial.ppt>.

"Kit #1023 - Fury Tempest 3D," Miniature Aircraft USA, Retrieved August 2007. <http://www.miniatureaircraftusa.com/support/kit_drawings.asp?kit='1023>.

"MAIN ROTOR BLADES FOR BIG HELICOPTERS," SAB Composites, June 2008. <http://www.sab-compositi.it/english/mainRotor.htm>.

"Nomadio Sensor 2". Nomadio R/C. Retrieved August 2005. <http://www.nomadio.net/default.asp?ilevel1=2>.

"Pololu USB 16-servo Controller," Pololu Robotics and Electronics, Retrieved June 2008. <http://www.pololu.com/products/pololu/0390/>.

"Spektrum RC DX3 Manual", Horizon Hobby, Retrieved August 2005. <http://www.spektrumrc.com/Media/PDF/dx3manual-english.pdf>.

"STARMAC Gallery," Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control. Retrieved July 2008. <http://hybrid.stanford.edu/starmac/gallery>.

"The JAUS Tutorial," Joint Architecture for Unmanned Systems, Retrieved June 2008. <http://www.jauswg.org/JAUStutorial.ppt>.

XCell Ion-X^2 Kit Details," Miniature Aircraft USA, Retrieved August 2007. <http://www.miniatureaircraftusa.com/helicopterkits/1024_Ionx/1024_kit_details.asp>.

Accardo, D.; Esposito, F.; Moccia , A.; "Low-cost avionics for autonomous navigation software/hardware testing", Aerospace Conference, 2004. Proceedings. 2004 IEEE, Volume 5, 6-13 March 2004.

Agent Oriented Software Pty. Ltd. (AOS), P.O Box 639, Carlton South, Victoria 3053. *JACK Intelligent Agents: JACK Manual*, 4.1 edition, April 2003. "http://www.agent-software.com/shared/resources/index.html"

Andrew Lucas, First Flight True Autonomy at Last. *Agent-Oriented Software Press Release*, July 2004.

Banks, C., "Helicopter Dynamic Stability," Retrieved June 2007. <http://www.aerojockey.com/papers/helicopter/report.html>.

Berg, M.; Kreveld, M.; Overmars, M.; Schwarzkopf, O.; *Computational Geometry*, Springer, 2000.

Berkeley Aerobot Team (BEAR), U.C. Berkeley, Retrieved October 2005. <http://robotics.eecs.berkeley.edu/bear>.

Buskey, G.; Wyeth, G.; Roberts, J.; "Autonomous helicopter hover using an artificial neural network", Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on Volume 2, 2001 Page(s):1635 - 1640 vol.2.

Bradski, G.R., Computer vision face tracking as a component of a perceptual user interface. In Workshop on Applications of Computer Vision, pages 214–219, Princeton, NJ, Oct. 1998.

Chang-Sun Yoo; Iee-Ki Ahn; “Low cost GPS/INS sensor fusion system for UAV navigation”, Digital Avionics Systems Conference, 2003. DASC '03. The 22nd, Volume 2, 12-16 Oct. 2003 Page(s):8.A.1 - 8.1-9 vol.2.

Erdem, U.M. and Sclaroff, S., “Automated camera layout to satisfy task-specific and floor plan-specific coverage requirements.” Computer Vision and Image Understanding, 2006, vol. 103 (3), pp. 156-169.

de Castro, E. and Morandi, C., "Registration of Translated and Rotated Images Using Finite Fourier Transforms", IEEE Transactions on pattern analysis and machine intelligence, Sept. 1987.

de Sousa, J.B.; Girard, A.R.; Hedrick, J.K.; “Elemental maneuvers and coordination structures for unmanned air vehicles”, Decision and Control, 2004. CDC. 43rd IEEE Conference, Volume 1, 14-17 Dec. 2004 Page(s):608 - 613 Vol.1.

Dijkstra, E.W., “A note on two problems in connexion with graphs. In Numerische Mathematik,” 1 (1959), S. 269–271.

Endurance R/C, “Products – PCTx,” Retrieved July 2008. <http://www.endurance-rc.com/pctx.html>.

Fregene, K.; Madhavan, R.; Kennedy, D.; “Coordinated control of multiple terrain mapping UGVs”, Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference, Apr 26-May 1, 2004 Page(s):4210 - 4215 Vol.4.

Fregene, K.; Kennedy, D.C.; Wang, D.W.L.; “Toward a systems- and control-oriented agent framework”, Systems, Man and Cybernetics, Part B, IEEE Transactions, Volume 35, Issue 5, Oct. 2005 Page(s):999 – 1012

Fregene, K.; Kennedy, D.; David Wang; “Multi-vehicle pursuit-evasion: an agent- based framework”, Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference, Volume 2, 14-19 Sept. 2003 Page(s):2707 - 2713 vol.2.

Fregene, K., “Distributed Intelligent Control of Hybrid Multiagent Systems”, Ph.D. thesis, University of Waterloo, Ontario, Canada, 2002.

Fregene, K.; Kennedy, D.; Wang, D.; “On the Stability of Coordinated Multiagent Systems with Degraded Communication”, *American Control Conference*, Volume 2, 4-6 June 2003 Page(s): 1038- 1043 Vol. 2.

Fregene, K.; Kennedy, D.; Wang, D.; “HICA: A Framework for Distributed Multiagent Control,” in *IASTED Int. Conf. on Intelligent Systems and Control* 2001, pp. 187-192

Gancet, J. and Lacroix, S.; “Embedding heterogeneous levels of decisional autonomy in multi-robot systems”, 7th International Symposium on Distributed Autonomous Robotic Systems, Toulouse (France), June 2004.

Gancet, J.; Hattenberger, G.; Alami, R.; Lacroix, S.; “Task Planning and Control for a multi-UAV system: architecture and algorithms”, IEEE International Conference on Intelligent Robots and Systems, Edmonton (Canada).

González, A.; Béjar, M.; Mahtani, R.; Ollero, A.;"Control and stability analysis of autonomous helicopters", International Symposium on Robotics and Applications (ISORA), World Automation Congress (WAC 2004), Seville (Spain), June 28 - July 1, 2004.

Hoffmann, G.; Rajnarayan, D.G.; Waslander, S.L.; Dostal, D.; Jang, J.S.; Tomlin, C.J.; “The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC)”, Digital Avionics Systems Conference, 2004. DASC 04. The 23rd, Volume 2, 24-28 Oct. 2004 Page(s):12.E.4 - 121-10 Vol.2.

Horizon Hobby, Inc., 4105 Fieldstone Road, Champaign, IL 61822 “Spektrum RC DX3 Manual”, Retrieved June 2006. <http://www.spektrumrc.com/Media/PDF/dx3manual-english.pdf>.

Hudson, T., “autopilot: UAV command and control,” SourceForge.net, Retrieved June, 2008. <http://sourceforge.net/projects/autopilot/>.

Jang, J.S., “Nonlinear control using discrete-time dynamic inversion under input saturation: Theory and Experiment on the Stanford DragonFly UAVs”, PhD thesis, Stanford University, 2003.

Jones, E.D.; Roberts, R.S.; Hsia, T.C.S.; “STOMP: a software architecture for the design and simulation of UAV-based sensor networks”, Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference, Volume 3, 14-19 Sept. 2003 Page(s):3321 - 3326 vol.3.

Kahn, A.D.; Foch, R.J.; “Attitude command attitude hold and stability augmentation systems for a small-scale helicopter UAV”, Digital Avionics Systems Conference, 2003. DASC '03. The 22nd, Volume 2, 12-16 Oct. 2003 Page(s):8.A.4 - 81-10 vol.2.

Kalman, R., "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME - Journal of Basic Engineering* Vol. 82: pp. 35-45 (1960).

Kanade, T.; Amidi, O.; Ke, Q.; “Real-time and 3D vision for autonomous small and micro air vehicles”, Decision and Control, 2004. CDC. 43rd IEEE Conference, 14-17 Dec. 2004 Page(s):1655 - 1662 Vol.2.

Karray, F.; Basir, O.; Song, I.; Li, H.; “A framework for coordinated control of multi-agent systems”, Intelligent Control, 2004. Proceedings of the 2004 IEEE International Symposium, 2004 Page(s):156 – 161

Karim, S.; Heinze, C.; Dunn, S.; “Agent-based mission management for a UAV”, Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004 14-17 Dec. 2004 Page(s):481 – 486.

Katwal, S., “Range Sensor Brick for Modular Robotics (project in lieu of thesis)”, Retrieved June 2008. <http://imaging.utk.edu/publications/papers/dissertation/skatwal_project.pdf>.

Kent, C. and Roberts, R., “Cooperation and path planning for unmanned air vehicles”, *Lawrence Livermore National Laboratory: Technical Report UCRL-JC-149915,* September 2002.

Koo, T.J.; “Hierarchical system architecture for multi-agent multi-modal systems”, Decision and Control, 2001. Proceedings of the 40th IEEE Conference, Volume 2, 4-7 Dec. 2001 Page(s):1509 - 1514 vol.2.

Kooshesh, A.A. and Moret, B.E., “Three-Coloring the Vertices of a Triangulated Simple Polygon,” Pattern Recognition, vol. 25, no. 4, pp. 443, 1992.

Krishnamurthy, K.; Ward, D.T.; “Intelligent systems for autonomous aircraft”, Systems, Man, and Cybernetics, 2000 IEEE International Conference, Volume 4, 8-11 Oct. 2000 Page(s):2369 - 2374 vol.4.

Krishnamurthy, K.; Ward, D.T.; “Intelligent systems for autonomous aircraft”, Systems, Man, and Cybernetics, 2000 IEEE International Conference, 8-11 Oct. 2000 Page(s):2369 - 2374 vol.4.

Lee, S.J.; Kim, S.P.; Kim, T.S.; Kim, H.K.; Lee, H.C.; “Development of autonomous flight control system for 50m unmanned airship”, Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004, 14-17 Dec. 2004 Page(s):457 – 461

Lemaire, T.; Alami, R.; Lacroix, S.; “A distributed tasks allocation scheme in multi-UAV context”, Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference, Volume 4, Apr 26-May 1, 2004 Page(s):3622 - 3627 Vol.4.

Lucas, B.D. and Kanade, T., An iterative image registration technique with an application to stereo vision. Proceedings of Imaging understanding workshop, 1981, pp 121—130.

Mahmoud, F.A.A., "Constructing & Simulating a Mathematical Model of Longitudinal Helicopter Flight Dynamics," Part of "Helicopter Control Design" on the MATLAB Central File Exchange, 2005, Retrieved June 2007. <http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=9222&objectType=file>.

Masar, I. and Gerke, M. "DSP-Based Control of Mobile Robots", *The European DSP Education and Research Symposium EDERS-2004*, Birmingham, UK, November 2004.

Masar, I. and Gabler, R., "An Integrated Environment for the Modeling, Simulation, and Rapid Design of Control Algorithms for Mobile Robots", MATLAB Digest, September 2005, Retrieved March 2006. <http://www.mathworks.com/company/newsletters/digest/2005/sept/mobile_robots.html>.

Merino, L.; Caballero, F.; Martínez-de Dios, J.R.; Ollero, A.; "Cooperative Fire Detection using Unmanned Aerial Vehicles", Proceedings of the 2005 IEEE, IEEE International Conference on Robotics and Automation, Barcelona (Spain), April 2005.

Naik, N., "Design, Development and Characterization of a Thermal Sensor Brick System for Modular Robotics," Retrieved June 2008. <http://imaging.utk.edu/publications/papers/dissertation/2006-aug-thesis-naik.pdf>.

Naik, N., "Infrared Imaging Sensor Brick for Modular Robotics (project in lieu of thesis)," Retrieved June 2008. <http://imaging.utk.edu/publications/papers/dissertation/nnaik_project.pdf>.

Nakanishi, H.; Inoue, K.; "Robust control system design by use of neural networks and its application to UAV flight control", Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on Volume 3, 25-29 July 2004 Page(s):1769 - 1774 vol.3

Ollero, A.; Lacroix, S.; Merino, L.; Gancet, J.; Wiklund, J.; Remuss, V.; Perez, I.V.; Gutierrez, L.G.; Viegas, D.X.; Benitez, M.A.G.; Mallet, A.; Alami, R.; Chatila, R.; Hommel, G.; Lechuga, F.J.C.; Arrue, B.C.; Ferruz, J.; Martinez-De Dios, J.R.; Caballero, F.; "Multiple eyes in the skies: architecture and perception issues in the COMETS unmanned air vehicles project", Robotics & Automation Magazine, IEEE, Volume 12, Issue 2, June 2005 Page(s):46 – 57.

Ollero, A.; Ferruz, J.; Caballero, F.; Hurtado, S.; Merino, L.; "Motion compensation and object detection for autonomous helicopter visual navigation in the COMETS system",

Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference, Volume 1, 2004 Page(s):19 - 24 Vol.1.

O'Rourke, J., *Art Gallery Theorems and Algorithms*, Oxford, New York, 1987.

Padfield, Gareth D. Helicopter Flight Dynamics: The Theory and Application of Flying Qualities and Simulation Modeling, AIAA Education Series, 1996.

Page, D.L.; Koshchan, A.F.; Chen, C.; Jackson, M.; Chang, C.; Abidi, M.A.; "Modular Sensor 'Bricks' and Unmanned Systems for Persistent Large Area Surveillance," EP&R and R&RS Topical Meeting, American Nuclear Society, pp. 137-144, Albuquerque, New Mexico, March 9-12, 2008.

Poduri, A., "Video Sensor Brick for Modular Robotics (project in lieu of thesis)," Retrieved June 2008. <http://imaging.utk.edu/publications/papers/dissertation/apoduri_project.pdf>.

Rathinam, S.; Zennaro, M.; Mak, T.; Sengupta, R.; "An architecture for UAV team control", Proc. IFAC Conference on Intelligence Autonomous Vehicles in Portugal, July 2004.

Rathinam, S.; Sengupta, R.; "A safe flight algorithm for unmanned aerial vehicles", Aerospace Conference, 2004. Proceedings. 2004 IEEE Volume 5, 6-13 March 2004.

Roberts, J.M.; Corke, P.I.; Buskey, G.; "Low-cost flight control system for a small autonomous helicopter", Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference, 14-19 Sept. 2003 Page(s):546 - 551 vol.1.

Rubin, I.; Behzad, A.; Huei-Jiun Ju; Zhang, R.; Huang, X.; Liu, Y.; Khalaf, R.; "Ad hoc wireless networks with mobile backbones", Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium, Volume 1, 5-8 Sept. 2004 Page(s):566 - 573 Vol.1.

Saphyroon, A.; Jarah, M.A.; Al-Ali, A.; Hadi, M.; "Design and implementation of a low cost UAV controller", Industrial Technology, 2004. IEEE ICIT '04. 2004 IEEE International Conference on Volume 3, Dec. 8-10, 2004 Page(s):1394 – 1397.

Sasiadek, J.Z.; Hartana, P.; "Sensor fusion for navigation of an autonomous unmanned aerial vehicle", Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on Volume 4, Apr 26-May 1, 2004 Page(s):4029 - 4034 Vol.4.

Sellers, D., "An Overview of Proportional plus Integral plus Derivative Control and Suggestions for Its Successful Application and Implementation," <http://www.peci.org/library/PECI_ControlOverview1_1002.pdf>.

Shoham, Y. and Tennenholtz, M., “On social laws for artificial agent societies: off-line design.”, *Artificial Intelligence*, vol. 74, no. 1-2, pp. 231-252, 1995.

Sinopoli, B.; Micheli, M.; Donato, G.; Koo, T.J.; “Vision based navigation for an unmanned aerial vehicle”, Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference, 2001 Page(s):1757 - 1764 vol.2.

Sousa, J.; Simsek, T.; Varaiya, P.; “Task planning and execution for UAV teams”, Decision and Control, 2004. CDC. 43rd IEEE Conference, 14-17 Dec. 2004 Page(s):3804 - 3810 Vol.4.

Teo, R.; Jung Soon Jang; Tomlin, C.J.; “Automated multiple UAV flight - the Stanford DragonFly UAV Program”, Decision and Control, 2004. CDC. 43rd IEEE Conference, Volume 4, 14-17 Dec. 2004 Page(s):4268 - 4273 Vol.4.

Timofeev, A.V.; Kolushev, F.A.; Bogdanov, A.A.; “Hybrid algorithms of multi- agent control of mobile robots”, Neural Networks, 1999. IJCNN '99. International Joint Conference, 10-16 July 1999 Page(s):4115 - 4118 vol.6.

Xsens Motion Technologies, “MT9 Inertial 3D Motion Tracker,” Retrieved June 2008. <http://www.xsens.com/download/MT9_brochure.pdf>.

Xsens Motion Technologies, “MT9-B Technical Documentation” MT9 SDK, Retrieved June 2008.
<http://www.xsens.com/index.php?mainmenu=support&submenu=downloads&subsubmenu=legacy>.

Yang Hui; Cheng Xhiping; Xu Shanjia; Wan Shisong; “An unmanned air vehicle (UAV) GPS location and navigation system”, Microwave and Millimeter Wave Technology Proceedings, 1998. ICMMT '98. 1998 International Conference on18- 20 Aug. 1998 Page(s):472 – 475.

Zhiqiang Wu; Kumar, H.; Davari, A.; “Performance evaluation of OFDM transmission in UAV wireless communication”, System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium, 20-22 March 2005 Page(s):6 – 10.

Ziegler, J.G. and Nichols, N.B., “Optimum settings for automatic controllers,” Trans. ASME, 1942.

Vita

Marcus James Jackson was born Smyrna, Tennessee on April 7th, 1982. His family moved to Murfreesboro, Tennessee where Marcus attended Homer Pittard Campus School through the sixth grade. He next attended Central Middle School for two years, followed by Oakland High School. He graduated Valedictorian in 2001.

Marcus began attending Tennessee Technological University in Cookeville, Tennessee in August 2001. Marcus was an active member and officer of the Sigma Phi Epsilon fraternity. In the Summer of 2004, he attended the Sigma Phi Epsilon national conclave in San Antonio, Texas as the chapter alternate delegate. He received his B.S. Computer Engineering from Tennessee Tech in May 2005, graduating Summa Cum Laude. While working on his undergraduate degree, Marcus interned at Consolidated Utility District in Murfreesboro for one Summer.

Marcus began working as a graduate research assistant for the Imaging, Robotics, and Intelligent Systems (IRIS) laboratory at the University of Tennessee, Knoxville, in August 2005. He will be graduating with an M.S. Electrical Engineering in August, 2008.

After graduation, Marcus will return to Murfreesboro to pursue a career in computer engineering in the Nashville area.