Rutgers University Autonomous Aircraft Team

Technical Report

2010 AUVSI UAS Competition

Amanda Gaetano, Anthony Garrison, Pat Hickey,
Stephen Indyk, Bradley Lord, Cogan Noll,
John Palmer, Adrien Perkins, Gregory Quinn,
Michael Varga, Taras Wallace, and David Wescott

Thank you to our sponsors:
Rutgers University Engineering Governance Council
Rutgers Alumni Association
WINLAB
Invensense, Inc.
ST Micro, Inc.
BP Hobbies

24th May 2010

1 Introduction

The Rutgers Autonomous Aircraft Team was formed in September 2008 with the goal of entering the 2009 Association for Unmanned Vehicle Systems International (AUVSI)[5] Unmanned Air Systems (UAS) competition[7]. We didn't make it to the 2009 competition after losing our aircraft only four weeks before. Our imaging system and autopilot system were also far from ready. In terms of NCAA sport eligibility, one could consider 2009 our 'red shirt' year. In 2010 we are still rookies.

The team is made of 12 Rutgers undergraduates:

Anthony Garrison, President	John Palmer
Amanda Gaetano, Vice President	Adrien Perkins
Pat Hickey, Electronics Team Leader	Michael Varga
Greg Quinn, Mechanical Team leader	Bradley Lord
Stephen Indyk	Taras Wallace
Cogan Noll	David Wescott

We are advised by Professor Tobias Rossmann, and several members of the Tri-County R/C Club [4]: Larry Kosinar, Bill Perkowski, Chris Eivan, and Mike Korb, amongst many who spent time mentoring us.

We have been generously funded by Rutgers Engineering Governance Council, Rutgers Alumni Association, WINLAB, a laboratory of the Rutgers University Department of Electrical Engineering, and BP Hobbies.

We would like to acknowledge Louis Stumpf, for donating the aircraft that became the *Daedalus*; Jeff Perkins, for his prolific networking and promotion of the team; Gregg Peters, of BP Hobbies, for getting us out of numerous binds; Joe Vanderveer, for keeping a roof over our heads; Dave Devries and Michael Maia, of Invensense, for their technical guidance and donation of a sensor; F. Pasolini of ST Micro, for donation of a sensor; Professor Chung-chieh Shan, for overseeing Pat's independent study on the IMU design; Ivan Seskar, for keeping us (and many other undergrads) in the black; and Roger Kondos and Brij Pathak, for contributing to the thermopile attitude sensor.

Figure 1: The recovery of our 2009 entry, the *Icarus*

2 Systems Engineering

2.1 Goals

For this entry we took a close look at our goals and requirements, and attempted to meet the "threshold" competition requirements successfully. We started the design by forming, as a team, some broad goals based on our experience in the 2009 competition. These goals allowed sub-teams and individuals to tackle the smaller problems while still keeping an eye on the big picture.

We also wanted to balance building components with buying components, keeping in mind our slim finances. While we couldn't build every part of our entry from scratch, we were motivated to build as much of our systems as possible for the learning experience. We tried to buy only components which we didn't have the time or resources to build ourselves.

- The air vehicle must be stable and slow, with ample payload capacity and 20 minute flight duration. It should be a proven and durable fixed-wing design.
- The autopilot must be based on a proven system, so we could focus on improving an existing system rather than reinventing the wheel. It also must be open-source, both because of the high cost commercial systems, and because we want to learn about and experiment with the internals of our autopilot system.
- The imaging system must be simple and capable. We assume the operator will be burdened with guiding the camera and identifying targets manually. We couldn't guarantee time and resources would permit us to add sensor information and control loops for camera guidance and computer vision for target identification.

We will be performing a manual take off and giving the autopilot system control of the aircraft once the plane is in the air. We will also perform a manual landing. We decided not to pursue the reach goals of autonomous takeoff and landing based on our inexperience and the risk of a crash during development.

2.2 System Design Decisions

2.2.1 Airframe System Engineering

Our team was divided by area of expertise into a mechanical team and an electrical team.The mechanical team began the year with the goal of producing two competition worthy airframes. These were to become the *Daedalus* and *Knight Two*.

One of the most difficult design decisions was to quantify "ample payload." Payload size and weight is the major determining factor in aircraft size. We originally assumed a 2.5lb electronics (flight computer, sensors, and batteries) payload and a 2.5lb pan-tilt camera 8" in diameter, requiring 6" of vertical space below the bottom of the aircraft. These considerations introduced two major design parameters: the aircraft must be capable of carrying a minimum 5lb payload, and must be situated to provide clearance for the pan tilt unit.

We also needed an easily controllable, stable platform to accommodate the autopilot, and we needed to keep flight speed low to make image capture as easy as possible. After consulting with R/C experts, a high wing trainer is looked like the most stable and proven fixed-wing design. Trainers traditionally have light wing loading for slow speed flight, and are able to carry a comparatively large payload.Their lower wing loading is easier to control, and easier to recover if power is lost.

Erring on the side of caution, we decided our aircraft should be capable of accommodating a worst-case scenario payload. We were, in part, cautious due to a previous error where we rebuilt an entire aircraft to accommodate a larger payload. We decided on a high wing trainer with a 10' to 12' wingspan, so that we could accommodate up to 5lbs of payload while maintaining suitable wing

loading. Though we managed to keep our payload weight well under 5lbs, the only major drawback to having an oversize aircraft is the difficulty of transport.

Our selection of a trainer type aircraft encouraged us to rethink our pan tilt design. We based our initial assumptions of an 8" diameter pan and tilt unit, mounted underneath the aircraft, based on a design that was to be mounted on the nose of our 2009 entry, an aircraft with a pusher motor configuration. When we considered the possibilities of either dramatically increasing the ride height of our aircraft, or making significant modifications to the width of the fuselage, we decided to redesign our pan tilt system to fit in a more suitable envelope.

2.2.2 Electronics System Engineering

The electronics team was in charge of the autopilot system and the imaging system. The biggest system design choice we made was to use a single board Linux-based flight computer as the platform for both systems. We felt that the best way to develop software which interfaces with a wide variety of analog and digital electrical systems, and has complicated information flow requirements.

We had sketched plans for an electronics system built entirely on a Linux-based flight computer as early as July 2009, but planned on using a more standard microcontroller-based autopilot system with an auxiliary Linux-based computer for the imaging system until December 2009. After burning out our microcontroller based autopilot for the second time, we were faced with a long lead time for an expensive replacement part. On that impetus, we redesigned our electronics systems to eliminate the microcontroller autopilot and ensure all of our replacement parts would be inexpensive and commercially available from many vendors. The move to a single Linux-based flight computer, while not without obstacles, was a successful design decision, as we'll describe in this paper.

2.2.3 Autopilot System Engineering

We felt the Paparazzi system was full featured and proven, but we weren't satisfied with the capabilities of the existing Paparazzi hardware. The "Tiny" v2.1[20], the latest autopilot hardware released by the Paparazzi project, is based on a 64 pin ARM microcontroller. Almost all of its input-output capabilities are dedicated to autopilot functions, making it difficult to expand functionality and share information (for instance, position and orientation estimates) with payload components. As previously stated, the Tiny is expensive, fragile[1] and often subject to long lead times.

In order to replace the functionality of the Tiny with a flight computer, we needed to select hardware which could provide the flight control software with information from the attitude sensor, GPS module, and long distance telemetry radio link. We also needed a servo output interface. The most important feature provided by the Tiny was the real-time processing and multiplexing of the safety pilot's radio control signals with the autopilot control signals. The flight computer's servo output hardware must manage this multiplexing functionality in a way that is not tied to the function of the flight computer itself.

2.2.4 Safety of Linux Based Flight Computer

We acknowledge that a Linux based flight computer does not support the requirements for hard real-time, deterministic computing that would be required of any real-world autonomous system. Given the resources available to the team (time, money, and manpower), such a fully assured system is out of reach.

We elected to take many steps in the software engineering to help ensure our flight computer will operate reliably. The most important of these was code review and incremental testing. Both Pat Hickey and Bradley Lord became intimately familiar with the source code and control flow inside

[1] In our own admittedly clumsy experience

the autopilot software, and all patches were reviewed and tested by both engineers. Numerous other safety measures are described in Section 4.2.

Because our entry is essentially an experimental system and will always be operated with a safety pilot present, we engineered the safety pilot override to operate independently from the flight computer. We have ensured that, short of losing power to the radio control receiver and servos, the safety pilot will always be able to take over with the flick of a single switch on his radio control transmitter. We have also ensured that the flight computer's servo controller will be able to detect a loss of computer control and servos to a recoverable default. Along with careful preflight testing, careful observation, and definitive analysis of any errors encountered during flight, these measures meet the spirit and letter of competition rules.

2.2.5 Imaging System Engineering

When designing the imaging system our major goals were to have 1) a camera able to capture imagery "up to $60°$ in all directions from vertically below the aircraft"[8], as per the rules; 2) a real-time video feed; 4) the ability to download pictures while in flight; and 3) the ability to remotely control the camera, particularly to trigger pictures.

It was difficult to find a camera to base our imaging system on which met all of these requirements. Many still cameras can be fitted on a pan tilt system to requirement 1, and provide a real-time "viewfinder" video feed to meet requirement 2. Requirement 3 could typically be managed by attaching the camera to a single board computer by USB cable, though usually the camera powers down when attached to the computer, and some sort of switch is needed to deactivate the USB cable once the data transfer is complete.

Requirement 4 had the greatest effect on our design decision. Typically, only high-end SLR-type still cameras provide a socket for remote control. These are beyond our price range, and are much larger and heavier than we would like.

We built a prototype using a Canon Powershot S3 IS [6], a camera which has no electrical remote control input. We soldered wires to the push-button pads for the shutter and zoom buttons so we could override them remotely. Those wires were attached to isolating transistors, and a microcontroller was used for control. After some time, this clever circumvention of the factory anti-static shielding resulted in a burnt out camera. We abandoned this scheme.

3 Flight Vehicle

At the time of this writing, we have completed and flown a single flight vehicle, which we have named the *Daedalus*. We have also nearly completed a very similar backup airframe which will serve as a backup in case the *Daedalus* has a disaster. The *Knight Two* will be fitted with nearly identical R/C flight electronics, and accept the same autopilot and imaging system payload as the *Daedalus*.

3.1 Airframe

3.1.1 Daedalus

Louis Stumpf, a member of our local R/C club, donated a 10ft 1in (3.07m) wingspan, 7ft 8in (2.33m) long, high wing trainer to our club. It met all of the requirements outlined previously.

The plane, first built out of foam and balsa from long-lost plans in 1986, was accepted graciously, but required quite a bit of work before it was once again flight-worthy. We removed the covering to find water damage and rot which resulted from years of storage. We stripped and rebuilt nearly the entire airframe, adding carbon-fiber reinforcements to the wing in anticipation of increased wing loading due to our payload. Once rebuilt and covered, the *Daedalus* weighed 19.5 lbs with engine. A full 24oz fuel tank, plus 4lbs electronics payload, would bring the maximum takeoff weight to 25lbs.

Figure 2: The *Daedalus*

We wanted to certify the wings and wing root would withstand large dynamic forces in flight. We designed a test to mimic the Federal Airworthiness Standards' structural requirements[2]. Our first test was designed to certify that the wing, supported at the wing root and fuselage, could take a bottom loading present in a $1.5g$ acceleration, with a safety factor of 1.5[3].

The plane was carefully supported under the center section, nose, and tail while pre-weighed bags of sand were slowly placed symmetrically and simultaneously on the underside of each wing, starting from the innermost rib. The *Daedalus*'s wings supported 64 lbs under static loading. This is 260% of the maximum take off weight, the force of a $1.5g$ acceleration multiplied by a 1.73 safety factor. We also rocked the plane ten degrees from side to side while fully loaded to simulate differential loads during flight. Our post-test structure inspection found no warped members or cracked joints.

A second test was designed to certify the wing could take a top loading present from a $1.5g$ acceleration with a safety factor of 1.5, and to certify that load could be transmitted to the ground through the fuselage and landing gear. With the aircraft fully assembled and sitting upright on the landing gear, we loaded the top of the wing along the quarter chord with 64 lbs of sandbags. As in our first test, we also rocked the plane 10° side to side to simulate uneven loading. Our post-test structure inspection once again found no warped members or cracked joints.

Satisfied, we made engine and surface tests (see Appendix A:) before taxiing the plane around a parking lot. We brought the plane back inside, examined it for signs of stress and, finding everything in good condition, deemed the *Daedalus* flight-worthy.

Figure 3: Load testing the underside of the *Daedalus*'s wings. Shown with 16lbs of load.

3.1.2 Knight Two

In order to mitigate a disaster such as we had last year, we decided to build another airframe which could replace the *Daedalus* at short notice. We selected an Aero Craft Ltd. 12 Foot Telemaster [1] because it was available as a balsa kit and similar in size to the *Daedalus*. The 12 Foot Telemaster has a 12ft 1in (3.68m) wingspan, and is 7ft 10in (2.38m) in length.We named it the *Knight Two* after the Rutgers University mascot, the Scarlet Knight.

At the time of this writing, the *Knight Two* is 85% complete. We plan to complete the aircraft and bring it to the competition as a spare. Since our testing plan for the *Daedalus* was successful, we will follow the same plan when the *Knight Two* is completed.

[2]Special Federal Aviation Regulation No. 23, Subpart C–Structure. Assuming full load on wing as described in 23.335 paragraph 4.i., where an aircraft must pull out of a 7.5° dive at cruise velocity and cruise power "with a load factor of 1.5 ($0.5g$. acceleration increment)" [19]

[3]As stated in 23.303 [19].

(a) The unfinished *Knight Two*

(b) A completed 12 Foot Telemaster. Photo Credit: Hobby Lobby

Figure 4: Knight Two

3.2 Power

3.2.1 Engine

Each airframe has a 45cc gasoline engine and 24 ounce gasoline tank. Based on our flight tests, with climbs to 400 feet AGL and a high throttle cruise, we use a little less than 1oz gasoline per minute. While actual fuel consumption may be less, we set a flight time cap at 20 minutes in order to always have a safety reserve of at least 4oz. We have tested to ensure that, within $\pm 10°$ of level flight, the fuel tank will feed the engine until less than 1/2oz fuel remains.

3.2.2 Batteries

Each airframe has nearly identical flight electronics. To power a Futaba 2.4GHz receiver and 6 servos for a 20 minute flight, we selected a 2 cell (7.2v) 3.2 amp hour Lithium Ion battery with a 5v, 10A capacity switching regulator made by Castle Creations. We use a low voltage monitor in-line with the battery, which beeps when cell voltage falls below 7.2v. The autopilot electronics are powered from a completely separate battery for safety reasons.

4 Autopilot

Our entry uses an autopilot system based off the open source Paparazzi project [28]. We ported the airborne code to Linux in order to use a single computer for all of our flight hardware and software.

4.1 Hardware

4.1.1 Single Board Computer

We selected the Beagleboard [14], a single board computer made by Texas Instruments. It features a 500MHz ARM Cortex A8 processor and 256MB ram. We installed Angstrom Linux [2] on an 8GB SD card. The Beagleboard has a few logic level serial, parallel, and general purpose IO interfaces on board, but for simplicity we chose to attach all of the devices through the USB bus. The Beagleboard provides one USB host port for this purpose, but the power supply can only sustain a low current draw. To provide more current to the attached devices, we use a Belkin 7-port powered USB hub.

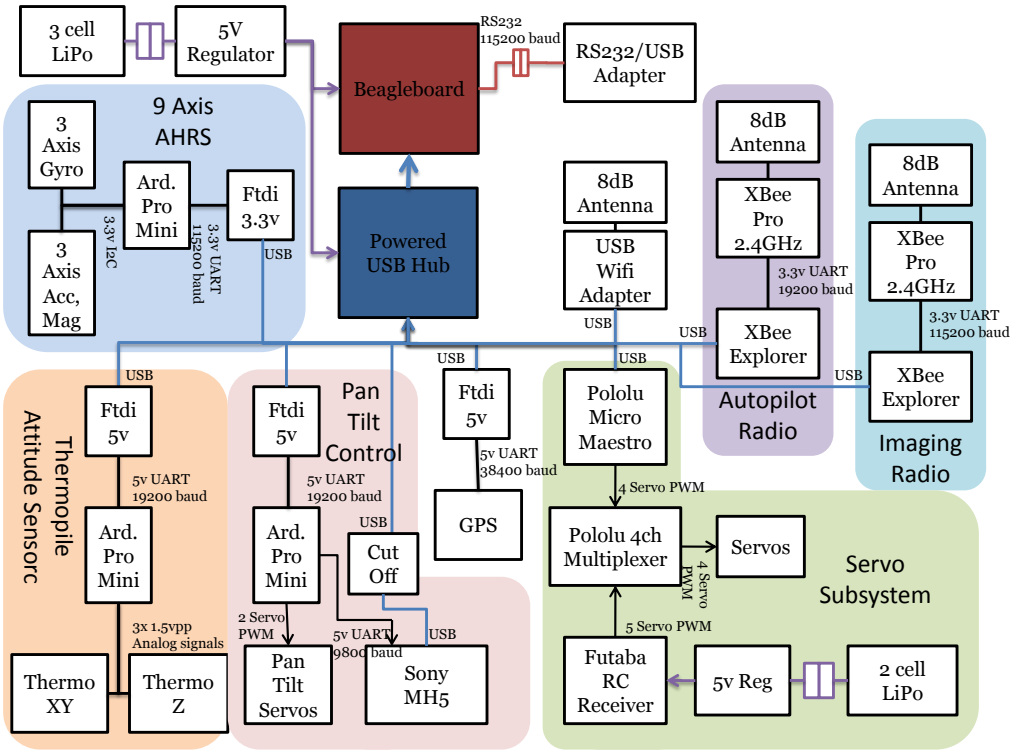


Figure 5: Hardware connection diagram for the autopilot and imaging systems. Only one of the 9 axis AHRS and Thermopile Attitude Sensor are used.

With a proper power supply, the USB hub will provide up to 500mA to each port on the 5v/GND circuit.

4.1.2 Servo Subsystem

We are using Pololu Micro Maestro [22], which communicates with the flight computer over USB and features 6 Servo PWM outputs. On Linux, the Micro Maestro enumerates as a `ttyACM` device. A simple serial protocol is used to update the servo PWM outputs. The Micro Maestro also features a configurable timeout and failsafe position for the servo PWM outputs. We have configured the device to bring each servo to a predefined position (throttle closed, elevator up, rudder right, ailerons left) after a 2 second new command timeout. This ensures that if the autopilot process or entire flight computer fails to respond for 2 seconds, the aircraft will enter this mode until the safety pilot takes control.

A Futaba 2.4GHz "FAAST" 6-channel radio transmitter and receiver [10] is the safety radio for our entry. We've tested the radio in the presence of our other 2.4GHz radios (Wifi adapter, autopilot radio, and imaging system radio) at a variety of ranges and configurations, and are confident that any radio interference is not harmful to the response of the servo outputs.

A Pololu 4 channel servo multiplexer [21] allows safety pilot is able to select whether the signals from the Futaba radio control receiver or the autopilot are in control of the servos. The four control channels (throttle, elevator, rudder, ailerons) are routed from both the Futaba receiver and Micro Maestro to the Master and Slave inputs, respectively, on the multiplexer. The multiplexer and servos are powered by the Futaba receiver, which has an independent power supply from the flight

computer. As long as the Futaba receiver has power and signal from the radio control transmitter, the safety pilot will be able to take control of the aircraft.

4.1.3 Attitude and Heading Reference Sensor

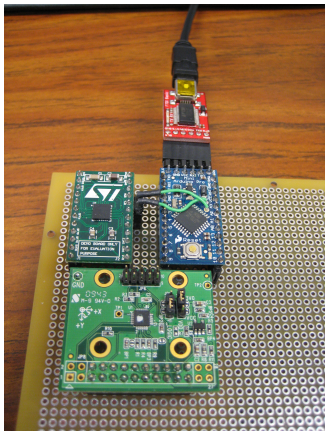


Figure 6: Completed AHRS sensor

Early in the school year, we built a thermopile-based attitude (pitch and roll) sensor very similar to the sensor design used by the Paparazzi Tiny autopilot. Accompanied by a driver which allowed it to appear to the airborne software just as it would on the Tiny, this simple analog sensor helped us prototype and test the Paparazzi system. Having built a superior sensor, we keep it as a backup. It is detailed in Figure 5 although we do not intend to use it for the competition.

Thanks to sensor donations from Invensense and ST Micro, we were able to build a 9 axis Attitude and Heading Reference Sensor (AHRS) from an Invensense ITG-3200 3-axis rate gyro [16] and an ST Micro LSM303DLH 3-axis accelerometer and magnetometer [25]. Both the ITG-3200 and LSM303DLH have an all digital interface, and communicate with a microcontroller on a single I2C [30] serial bus. The microcontroller performs the self-test on the LSM303DLH at start-up, and proceeds to send uncalibrated samples to the flight computer over its UART connection. Samples of all 9 axes (three each of gyro, accelerometer, and magnetometer) are taken and sent to the flight computer at 40Hz.

The two sensors and an AVR microcontroller, mounted on breakout boards, were hooked up using simple 0.1" pitch protoboard, as shown in Figure 6. The sensor board is powered by the Sparkfun FTDI serial to USB adapter [12] the microcontroller uses to communicate with the flight computer.

4.1.4 Other Components

- GPS: We selected the GS407 [11], a uBlox LEA-5H [27] based GPS with an active helical antenna. We picked the unit because the uBlox chipset has been used successfully with Paparazzi in the past, and provides an internal Kalman filter mode for aircraft expecting accelerations of $\pm 2g$. The GPS is powered and connected to the USB hub with a Sparkfun FTDI serial to USB adapter [12].
- Autopilot Radio: We selected a Maxstream XBee Pro (Series 2.5) 2.4GHz 50mW [18] radio system for the autopilot telemetry and uplink radio. It operates at the relatively slow baud rate of 19200bps. We selected this system because it is known to work well with the Paparazzi system, and it can provide a seamless point-to-point UART pipe with two transceivers. Importantly, it deals well with interference from other 2.4GHz radio connections nearby, including other XBee Pro radios, the Futaba "FAAST" radio control signal, and 2.4GHz 802.11g Wifi.
- Imaging Radio: We are using an identical pair of XBee Pro radios in a point-to-point 115200bps UART connection configuration to connect the imaging system to the ground. We have tested and found that, when operating two point-to-point connections at once, neither suffers measurably at up to 1/3 mile line of sight range. Therefore, rather than write complicated software to use both the autopilot serial communication protocol and the imaging radio serial communication protocol with the same XBee radio, we use two radio connections on the same band with little penalty.
- Wifi Adapter: We use a 802.11g (wifi) adapter for convenience in the lab and at the field to program the flight computer and supervise the execution of autopilot process. The reliability

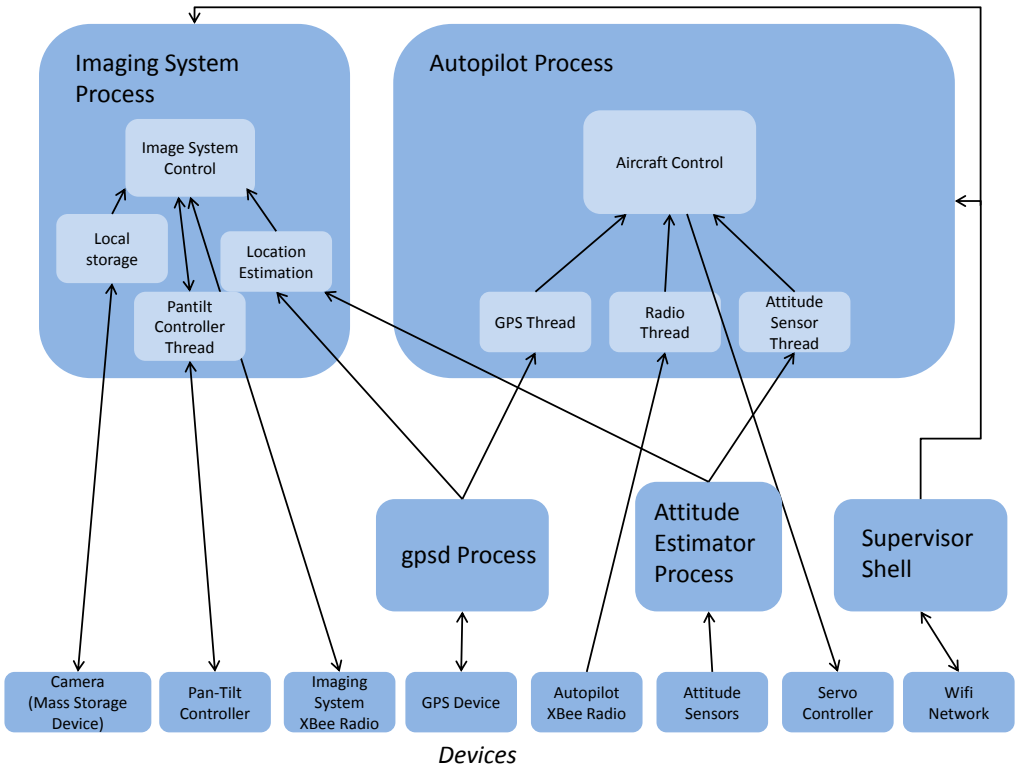


Figure 7: Software communications diagram

of our wifi connection drops off significantly beyond 600ft (200m), so we do not actively use the link during flight or for any telemetry.

4.1.5 Power

Both the Beagleboard and the powered USB hub are provided with 5VDC via a switching regulator and a 3 cell (11.1v nominal) 1.5Ah Lithium Polymer battery. This provides, on average, 90 minutes of operation with all devices on.

4.2 Autopilot Software

The autopilot software is a port of the Paparazzi airborne software to Linux. The airborne software is written in C and originally assumed a bare-metal (that is, no operating system) microcontroller environment with memory-mapped IO. The port mostly constituted inserting several layers between blocking Linux system calls and the control loop's expectation of non-blocking memory reads. The simplest way to do this was by splitting the autopilot process into separate threads for aircraft control and device access. We use mutexed shared memory to communicate between these threads, which works well because the aircraft control thread is a loop which only runs (therefore, only needs a lock) 60 times per second.

The multithread approach helps ensure that the aircraft control thread will not be disrupted by busy waiting. It results in a simple structure of the whole program, and minimized the amount of code we had to change in the port. We care about minimizing changes to Paparazzi because the code is known to be well tested and reliable.

The multithread approach works well for resources which are only needed by the autopilot process, but both the attitude and GPS information are needed by both the autopilot and imaging system processes. To remedy this, we use `gpsd` [3], the standard Linux GPS service, to interface with the uBlox GPS device over a serial port. `libgps`, a library for interfacing with `gpsd`, provides updates to client threads in both the autopilot and imaging processes. We handle the AHRS updates in the same way, except instead of the specialized `libgps` library we use the generic `mqueue` [17] service, a message passing facility provided by the Linux kernel. Each client thread opens a named queue, and the AHRS process delivers updated measurements to each queue.

4.3 Attitude and Heading Reference Sensor Software

The physical AHRS board sends 9 uncalibrated measurements to the flight computer, but the aircraft control loop is requires in attitude and heading expressed as 3 Euler angles and their derivatives with respect to time. It is straightforward to estimate Euler angle derivatives from rate gyro measurements, but estimating three degrees of angular orientation from six magnetometer and accelerometer measurements is a difficult nonlinear operation.

While their discussion is outside the scope of this paper, many algorithms exist for this conversion. We use a Quaternion Kalman Filter algorithm described in [9], implemented as a complete filter in Haskell. Angular orientation is tracked in quaternion form, and translated into euler angles for the use of the autopilot and imaging processes. This implementation was carried out as part of an independent study by team member Pat Hickey, and more details can be found in [13].

4.4 Ground Station

Because we based our autopilot software on the Paparazzi project, we were able to make use of their excellent ground control software (GCS) with essentially no modifications. The Paparazzi GCS provides a map interface that allows the operator and competition judges to inspect the aircraft's position to ensure it is within the allowed airspace. It also provides attitude and altitude telemetry, and notification of lost radio contact, so the operator can alert the safety pilot to unsafe operation.

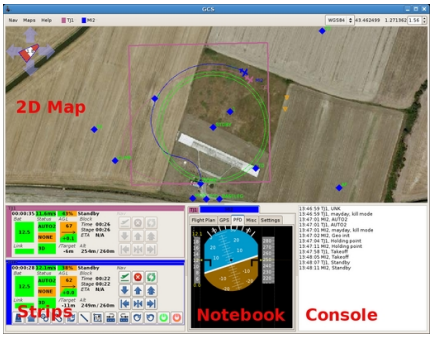


Figure 8: Paparazzi GCS. Image Credit: Paparazzi Wiki [**?**]

5 Imaging System

To obtain images of targets in flight we have created an Imaging System capable of meeting all the requirements of the competition. The system consists of a pan-tilt camera on the plane, communication software on the plane, and our "Image Station" application running on a dedicated laptop. The Image Station communicates with the plane in real time via two different sets of wireless radios. Using the Image Station, the operator can view a live video stream from the plane, control the pan-tilt camera remotely, and download and manipulate images.

5.1 Camera

We selected the Sony MHS-PM5 camera as the basis of our imaging system. Marketed as a pocket 1080i video camera capable of 5 megapixel still images, we primarily selected the camera because it features a swiveling lens at one end of the body and it features Sony's LANC (Local Application Control Bus System) [29]. Because of the swiveling lens we were able to achieve two rotational degrees of freedom while only rotating the body of the camera about one axis. This greatly simplified the

(a) Front View (b) Isometric View

Figure 9: Sony MHS-PM5. Image Credit: Sony

pan-tilt design, and also allowed us to easily mount the camera primarily inside the fuselage of the aircraft, which protects it from damage during takeoff and landing, and reduces drag on the airframe.

The LANC protocol is based on a 9600 baud UART serial connection, and is available on the PM5's 10-pin "Multi-AV" connector. Camera control commands, which include commands to zoom in, zoom out, and take a still picture, are publicly documented. Also on the "Multi-AV" connector are the signals for NTSC video and audio output. We use both the NTSC output and LANC signals on this connector using a LANC Adapter cable [26].

The PM5's NTSC video signal outputs a 480 line video of the current "viewfinder", the live video that is also displayed on the rear LCD screen. Still pictures are taken based on targets spotted in this video feed and are stored on the camera's memory card.

Each still picture is 5 megapixels, with an actual resolution of 2592 x 1944. To determine how the images will appear from different altitudes, we have created a table that maps the size of features in the world to pixels in our images at different altitudes. We also take into consideration the changes in distance to the ground when the camera is tilted at a (° 45) angle as opposed to vertically aligned with the plane. We can see that at an altitude of 100 ft. each pixel will represent an area a little over 1/3 of an inch wide when looking straight down, which should be a sufficient level of detail to accurately identify any target. However, at an altitude of 500 ft. each pixel will represent an area almost 2 inches wide when looking straight down. We are confident that this level of detail will be enough to identify the shapes and colors of all targets at any required altitude, but we may have some difficulty identifying alpha-numerics on very small targets at high altitudes. As a first year team, we feel that meeting the "threshold" rather than the "objective" goals, at the edge of the altitude envelope, is an acceptable level of performance.

Altitude (ft)	**Pixel width at 0° (in)**	**Pixel width at 45° (in)**
100	0.38	0.54
200	0.76	1.08
300	1.14	1.62
400	1.52	2.16
500	1.90	2.69

Figure 10: Linear resolution per pixel as a function of altitude, at directly below flight path and 45° off flight path

5.2 Pan Tilt Unit

In order to meet the target imaging requirments we needed to add both pan and tilt actuators to the camera. The field of view of our lens is only $45°$ by $34°$, which is not enough to meet the competition field of view requirements of $60°$ vision in each direction. To address this issue, we designed a pan tilt unit which makes it possible to image the entire hemisphere below the fuselage of the aircraft.

We took advantage of the MHS-PM5's swiveling lens assembly (see Figure 9a, and added a servo actuator which can rotate the lens $135°$ back from straight forward, as shown in Figure 12a. The tilt servo is a Hitech HS 77BB [23], and is capable of $180°$ rotation. Figures 11 and 12a show the camera lens tilted at $45°$ below the horizon. The camera and tilt servo assembly is attached to the aircraft on a rotating mount. A thrust bearing takes lateral loads off the pan servo shaft, and stabilizes the assembly. The pan servo is a Hitech HS 785HB [24], and is capable of $1260°$ rotation (3.5 revolutions). Both servos are powered and controlled by the Pan-tilt control board.

The total height of the imaging system is 6.75", a maximum 2" of which is exposed below the fuselage of the aircraft. The rotating assembly is, on its widest dimension 2.13", and fits inside a 2.5" cylinder. Wires exiting the side of the camera, inside the fuselage, make the assembly a bit wider. Because the maximum rotation is only $1260°$, the camera and tilt servo wires only have to be capable of wrapping $\pm 630°$ around the shaft below the thrust bearing. A wire guide hass been added to keep the wires from tangling.

The pan servo is calibrated before installation by mapping servo PWM inputs to a measured angular output at $30°$ intervals. Linear interpolation is used to determine PWM outputs which achieve intermediate angles. The tilt servo is calibrated at $10°$ intervals, due to its much smaller rotation range. As shown in Figure 12b, the angular acceleration of the camera angle linkage (Alpha4) is almost constant compared to the servo arm linkage throughout the range $[0°, 135°]$. The linkage has been designed to keep the majority of nonlinear rotation outside the camera's range of rotation. Therefore, we can consider the calibrated servo output to be directly related to the camera angle. The endpoints of the tilt rotation rang are calibrated and enforced in software, because pushing the camera outside this range may damage it.

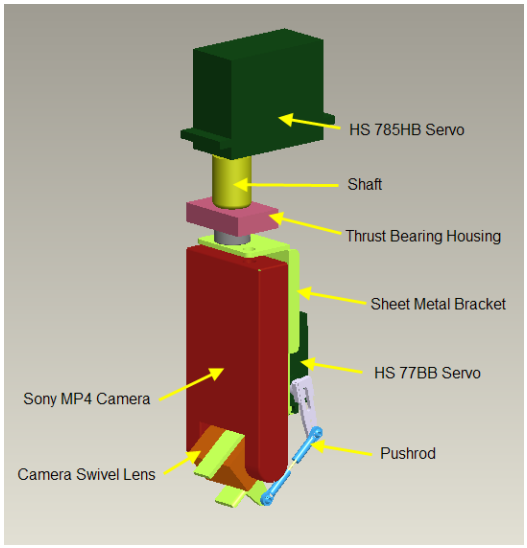
5.3 Obtaining Images

The entire imaging system is controlled by the "Image Station Operator". They view a live video stream coming from the camera on the plane, which is transmitted over a high bandwidth radio. The operator can also control the camera by sending commands to the plane over the XBee radio link. In order to assist the operator in finding targets, a target locating algorithm will be run on the video feed to highlight possible targets. At the time of writing work is in progress on this feature, and it is being implemented using the OpenCV library [15] with contour detection and polygonal mapping functions.

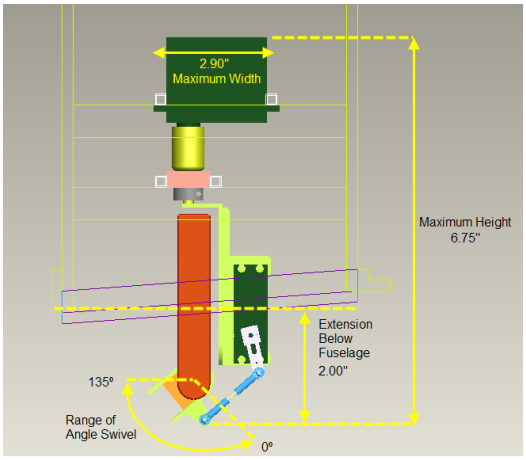
When a picture is taken on the plane it is immediately saved on the camera's memory card. In order to view that picture on the ground it must first be transferred to the plane's main computer (beagle board). This is done via the PM5's USB connector. When attached to a computer, the PM5 powers the camera off and appears as a USB Mass Storage Device. In order to use the PM5 as both a camera and Mass Storage Device in flight, we have spliced a transistor into the power line on the USB cable. This controls whether the camera detects it is connected via USB by blocking or allowing current in that line.

5.4 Downloading Images in Real Time

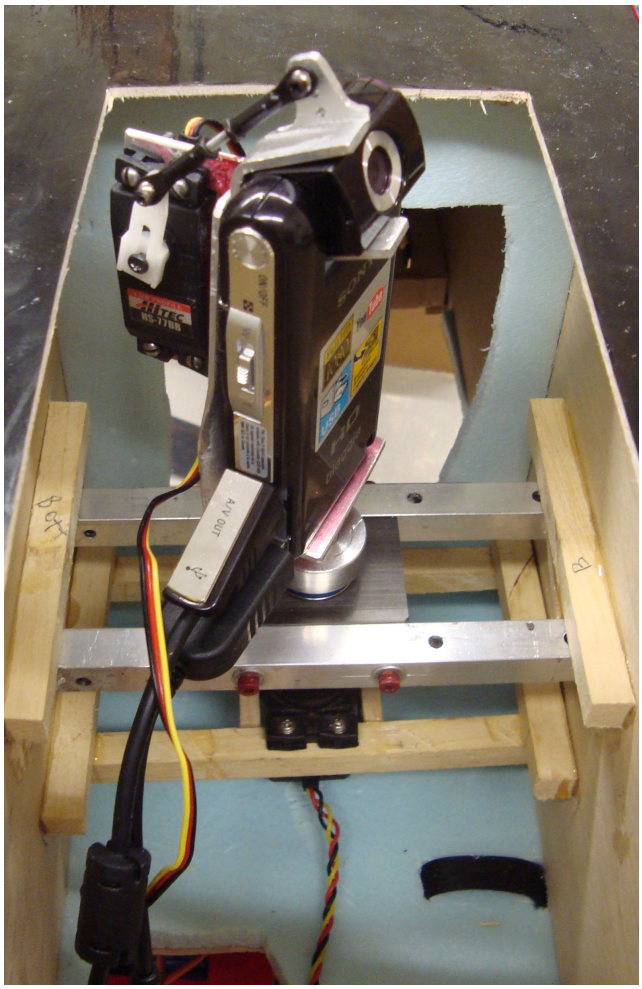
Once images are on the plane's computer they are ready to be downloaded over a 2.4GHz XBee radio link. Each picture taken by the camera is about 500KB in size, while our theoretical maximum throughput is only 115200 baud (and only about half of that in practice). Each full image takes



(a) Isometric View

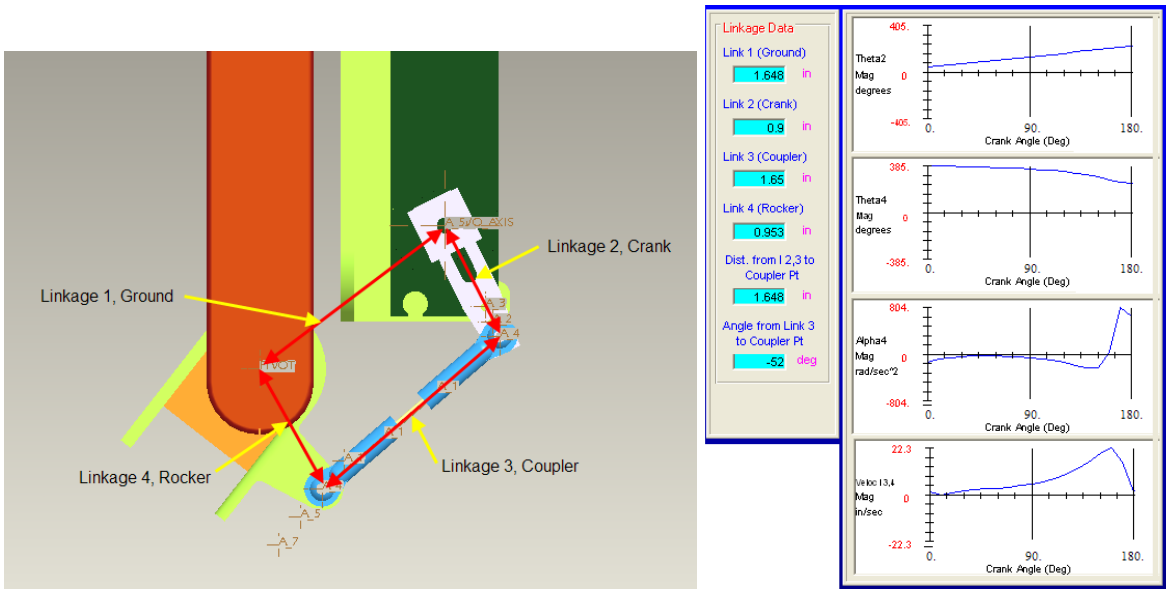


(b) Side View.



(c) Photograph of pan tilt unit installed in *Daedalus*

Figure 11: Pan Tilt Unit



(a) Linkage Detail View

(b) Linkage Velocity Plots

Figure 12: Tilt Linkage Detail

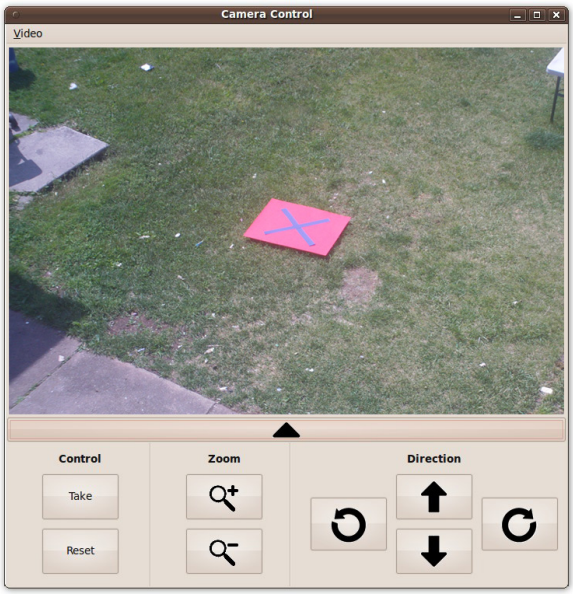


Figure 13: The Control Panel for the Image Station. Above is a live video feed from the plane. Below are controls to adjust the pan/tilt of the unit, zoom in/out, take a picture, and reset the camera to a neutral position.

almost three minutes to download, so we created a method to speed up the process. The entire image is first downloaded as an 800x600 low resolution 'thumbnail.' Once the thumbnail is available for viewing, the operator can select an area of interest and download a high-res crop of just that area. Downloading both the crop and the thumbnail generally takes under 30 seconds, less than 1/5 the time of downloading the full image, allowing us to quickly obtain high resolution images of the targets of interest.

	Size (KB)	Download Time (s)
Original Image	1200	166.7
Thumbnail	125	17.4
Crop	25	3.5

Figure 14: Size and download time for a full size image vs. a low-resolution thumbnail and a cropped area

Below a diagram is included that shows the entire flow of events from start to finish in our Imaging System for capturing an image and then downloading it.

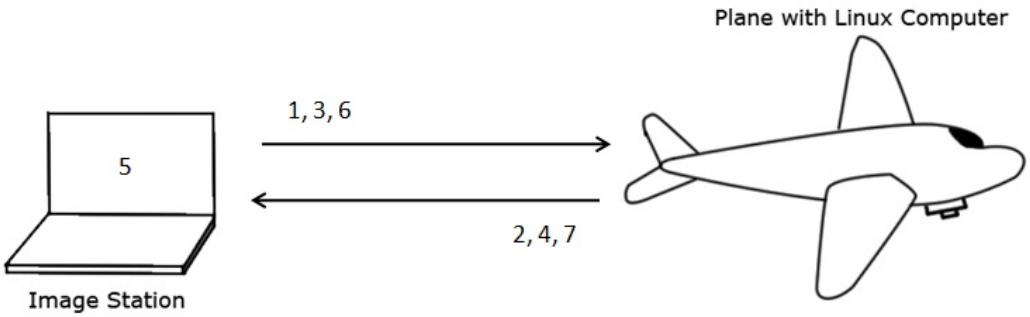


Figure 15: Flow of events for capturing an image

1. Command to take a picture is sent to Plane from Image Station
2. Plane sends command to camera, picture is taken and acknowledgement is sent to Image Station
3. Operator issues command to make images available for download
4. Plane acknowledges request and begins transfer of images from internal camera memory to the BeagleBoard. Plane sends notification when transfer is complete.
5. Newly available image(s) are automatically put on a queue to be downloaded.
6. A request to download a 250 byte chunk of the image at the top of the queue is sent to the plane.
7. The plane sends down the requested chunk of image.
8. Steps 6-7 are repeated throughout the duration of the flight. These image downloading commands are low priority and will only be sent if there are no other commands to be executed. This allows for high priority commands like taking a picture or moving the camera to be executed at any time.

5.5 Identifying target characteristics

When an image has been fully downloaded to the ground targets can be identified and tagged with metadata. Information about the plane's orientation, gps coordinates, altitude, yaw/pitch/roll, and pan/tilt of the camera are sent down when the image is downloaded. Using this information the GPS location of the target can be determined using the intrinsic properties of the camera and some basic trigonometry. Target shape, color, alphanumeric, alphanumeric color, and orientation are all entered manually by the operator.

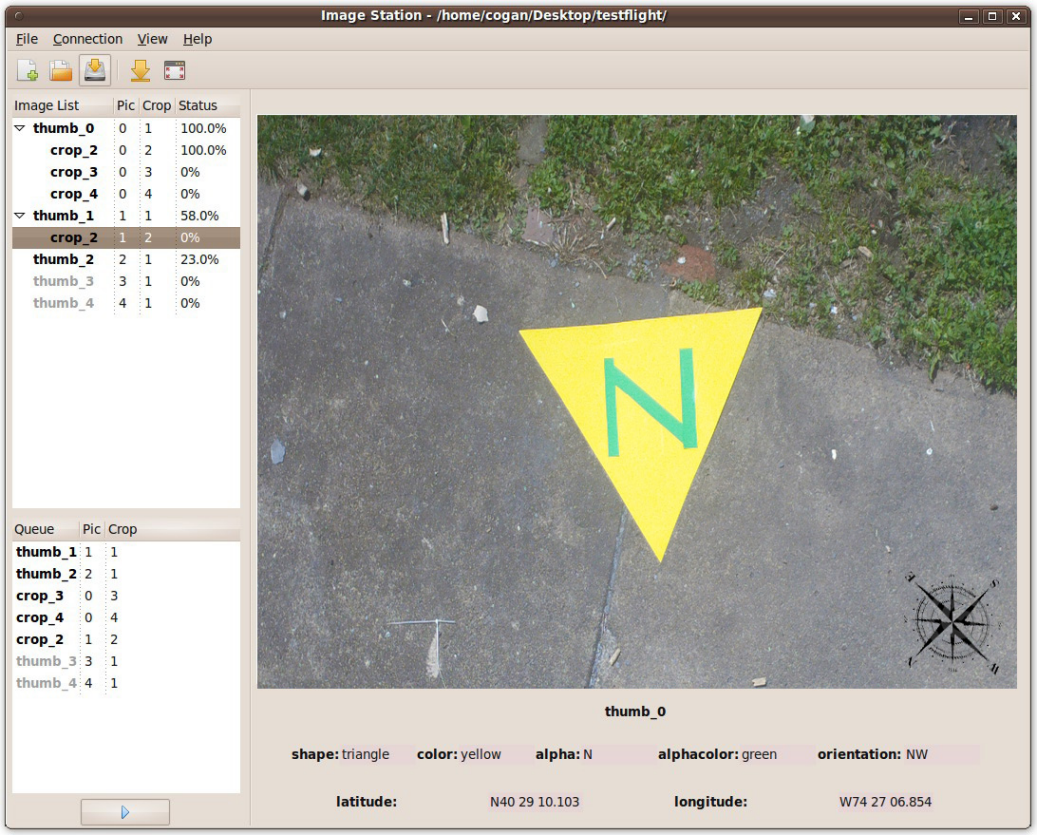


Figure 16: The main interface for the Image Station. All the images that have been taken are displayed on the top left. The queue of images to download is on the bottom left. On the right is the currently displayed image, with target information underneath.

6 Operation Safety

We consider the safety of our team, our aircraft, and our spectators to be of utmost importance. As such, we've implemented numerous safety features and developed a strict procedure for starting, taxiing, and flying the plane. This procedure enables us to ensure that all personnel are safely in place, and that no integral checks are missed.

6.1 Safety Features

Our system is designed to gracefully handle any failure while continuing to ensure the safety of those nearby.

- The channel 5 toggle switch determines whether the radio control transmitter, or the autopilot, has control over the flight control servos. The radio control safety pilot may disengage the autopilot at any time by flicking this switch.
- If the plane loses telemetry with the ground station for more than 10 seconds, the autopilot will default to a "return home mode" that will direct the plane towards the ground station in hopes of reestablishing a connection.
- If the connection telemetry remains dead for 30 seconds, the autopilot will bring the servos to a predefined default (throttle closed, elevator up, rudder right, ailerons left). The servos will remain in that position until the safety pilot takes control of the aircraft.
- If the flight computer does not provide the servo controller with a new servo output value for longer than 2 seconds, the servo controller will bring the servos to a predefined default (throttle closed, elevator up, rudder right, ailerons left). The servos will remain in that position until the safety pilot takes control of the aircraft.
- During engine start, the centerman (see Appendix A:) may kill the engine at any time with the ignition switch located on the right side of the aircraft, behind the engine mount.
- Radio control channel 6 functions as a remote kill switch on the *Knight Two*, disengaging the electronic ignition system of the engine. (The *Daedalus* has a magneto ignition, so we only have the manual kill switch behind the engine mount)

6.2 Procedures

Our team employs a clearly defined flight routine every time the engine is started. This routine ensures that everyone in the starting area is fully alert and aware at all times, that no one stands in a position of known danger (i.e. the prop arc), and that all know when the starter makes contact with the spinner. A set script with clear key words not only ensures the safety of our team, but also is conducive to efficiency, consistency, and repeatability.

6.3 Safety Observers

A safety observer is defined as a person, devoid of any additional specific responsibilities, who remains in the starting area during all pre-flight procedures. The safety observer is expected to watch for unsafe conditions, included but not limited to non-essential personnel in the starting, takeoff, and/or flight areas; loose mechanical connections; failed surface and/or range checks; and other forms of mechanical failure. The safety observer has the absolute authority to give the order to kill the ignition and abort startup immediately upon observing any unsafe condition. The safety observer is responsible for communicating possible conditions to the startup team as they arise. At minimum, one safety observer is required to start the engine, though we typically have more.

Appendix A: Preflight Procedures

1. Assemble plane.
2. Check all mechanical and electrical connections.
   - Are all nuts/bolts loc-tited?
   - Are all servo extensions connected and properly secured?
   - Are all access hatches secured?
3. Position wingmen[4], centerman[5].
4. Pilot turns on transmitter, announces "transmitter on."
5. Pilot requests "receiver on". Centerman turns on receiver, acknowledges.
6. Pilot completes surface checks:
   - Ailerons
   - Elevator
   - Rudder
   - Throttle
7. Starter primes engine.
8. Pilot obtains visual and verbal confirmation that wingmen, centerman are in place and alert.
9. Pilot requests "ignition on." Centerman moves ignition switch to the "on" position and acknowledges.
10. Starter requests "prop clear." Starter obtains visual confirmation that no person is standing in the prop arc and announces "prop clear."
11. Starter announces "contact" and starts the engine.
12. Starter clears the area.
13. Pilot repeats surface and throttle checks at close range.
14. Pilot leaves starting area and performs surface and throttle checks from a distance (range test).
15. Pilot returns to starting area to prepare for taxi.
16. Pilot requests "centerman off." Centerman carefully clears the area and acknowledges.
17. Pilot requests "wingmen off." Wingmen carefully clear the area and acknowledge.
18. Pilot taxis airplane to appropriate location to begin takeoff.
19. Pilot commences takeoff.

References

[1] Aero Craft Ltd. http://www.aerocraftrc.com/.

[2] Angstrom Linux Distribution. http://www.angstrom-distribution.org/.

[3] gpsd, a GPS service daemon. http://gpsd.berlios.de/.

[4] Tri-County RC Club of NJ. http://www.tricountyrc.com.

[5] AUVSI. Association for Unmanned Vehicle Systems Iinternational. http://www.auvsi.org.

[6] Canon. Powershot S3 IS. http://www.dpreview.com/news/0602/06022111canons3is.asp.

[7] AUVSI Seafarer's Chapter. 2010 Student Unmanned Aerial Systems competition. http://65.210.16.57/studentcomp2010/default.html.

[4] Two men, one positioned at each wing tip, preventing the plane from moving during engine start and subsequent surface checks.

[5] Man straddling the fuse, preventing the plane from moving during engine start and subsequent surface checks

[8] AUVSI Seafarer's Chapter. 2010 Student Unmanned Aerial Systems competition rules. http://65.210.16.57/studentcomp2010/rules/2010%20RFP20090824.pdf.

[9] D. Choukroun, I. Y. Bar-Itzhack, and Y. Oshman. A Novel Quaternion Kalman Filter. *TAE*, (930), 2004.

[10] Futaba Corporation. FAAST 2.4GHz Radio Systems. http://www.futaba-rc.com/.

[11] Sparkfun Electronics. 50 channel GS407 helical GPS receiver. http://www.sparkfun.com/commerce/product_info.php?products_id=9436.

[12] Sparkfun Electronics. FTDI basic breakout. http://www.sparkfun.com/commerce/product_info.php?products_id=8772.

[13] Patrick Hickey. Design of an Attitude and Heading Reference Sensor. http://moreproductive.org/hs-qkf-paper.pdf.

[14] Texas Instruments. Beagleboard. http://beagleboard.org/.

[15] Intel. Open Source Computer Vision Library. http://opencv.willowgarage.com.

[16] Inc. Invensense. ITG-3200. http://invensense.com/mems/gyro/itg3200.html.

[17] Linux manpages. man 7 mq_overview. http://man-wiki.net/index.php/7:mqoverview.

[18] Maxstream. XBee Pro Series 2.5 2.4RF modules. http://www.digi.com/products/wireless/zigbee-mesh/xbee-digimesh-2-4.jsp#overview.

[19] Code of Federal Regulations. Title 14. Aeronautics and Space. Special Federal Aviation Regulation no. 23. http://www.faa.gov/aircraft/air_cert/airworthiness_certification/std_awcert/std_awcert_regs/regs/.

[20] Paparazzi Project. Tiny v2.1 autopilot. http://paparazzi.enac.fr/wiki/Tiny_v2.

[21] Pololu Robotics and Electronics. 4 Servo Multiplexer. http://www.pololu.com/catalog/product/721.

[22] Pololu Robotics and Electronics. Micro Maestro. http://www.pololu.com/catalog/product/1351.

[23] ServoCity. HS-77BB Low Profile. http://www.servocity.com/html/hs-77bb_low_profile.html.

[24] ServoCity. HS-785HB 3.5 Rotations. http://www.servocity.com/html/hs-785hb_3_5_rotations.htmll.

[25] Inc. ST Micro. LSM303DLH. http://www.st.com/stonline/products/families/sensors/motion_sensors/lsm303dlh.htm.

[26] Studio1Productions. LANC Adapter for Sony Video Cameras. http://studio1productions.com/lanc-sa.htm.

[27] u blox. LEA-5H GPS receiver module. http://www.u-blox.com/en/lea-5h.html.

[28] ENAC University. Paparazzi Project. http://paparazzi.enac.fr.

[29] Versatile Video. How SONY's LANC™protocol works. http://www.boehmel.de/lanc.htm.

[30] Wikipedia. I2C. http://en.wikipedia.org/wiki/I2C.

The LaTeX source and figures of this paper are available on github:
http://github.com/pchickey/ruaiaa-2010doc/