AUTOMATED FLIGHT CONTROL OF AN UNMANNED BLIMP

NSF Summer Undergraduate Fellowship in Sensor Technologies
(Supported in part by Microsoft Corporation)
Vito Sabella, Electrical Engineering – University of Pennsylvania
Advisor: Dr. James Ostrowski and Dr. Jorge Santiago.

ABSTRACT

For the 2001 SUNFEST Research Program I am using sensor fusion of GPS, and rate gyroscope systems to automate the flight control and measure the dynamics of a 30 foot unmanned blimp. This project is parented by Professor James Ostrowski of the University of Pennsylvania GRASP Lab.

Using an onboard computer, a Garmin GPS36 GPS sensor, a pair of rate gyroscopes with supporting hardware, I developed a custom operating system in Java as well as sensor integration software for this operating system. I have developed a system to measure blimp dynamics while it flies and created a set of mathematical models to describe the blimp.

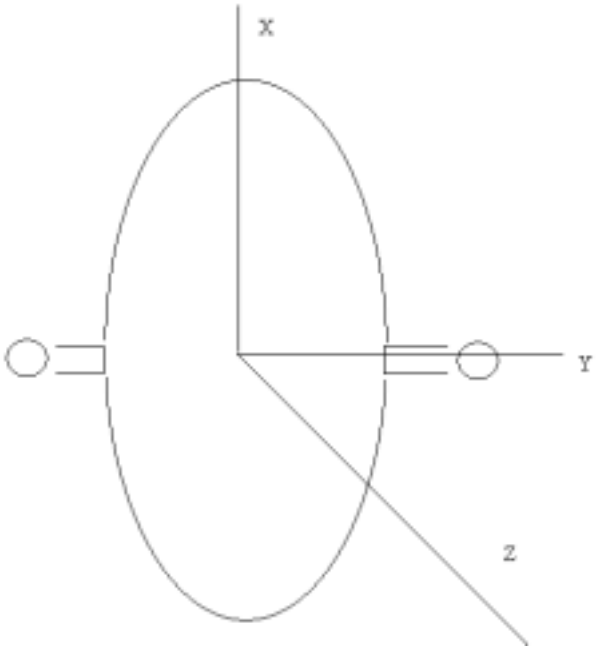
This project is sponsored by the National Science Foundation, the University of Pennsylvania School of Engineering and Applied Sciences, the University of Pennsylvania Science and Technology Wing living-learning program, and the University of Pennsylvania College House System.

1. Description of the project:

1.1 Definitions and Mathematical Model Characteristics

The Blimp Project is designed to automate the flight control of an unmanned 30-foot airship in order to provide a stable sensors platform for further research. Because of the size of our blimp – too large to be an indoor-only electric blimp and too large to be a person-carrying dirigible with powerful wind fighting engines – designing a system to maintain the vehicle's stability outdoors throughout varying winds is rather difficult.

The first step of designing our system was to create a model that accurately reflects the characteristics of the blimp. In order to limit our Euler-Cauchy equations to manageable dimensions we decided to eliminate roll as a factor that would require feedback for control. Utilizing a blimp-fixed coordinate system, the X-axis is motion in the forward direction of the blimp, the Y-axis is motion perpendicular to the sides of the blimp and parallel to the engine arms, and the Z-axis is motion perpendicular to the blimp and the engine arms.



Pitch is defined as rotation around the Y-axis, and yaw is rotation about the Z-axis. Roll, which is rotation about the X-axis, is negated.

Various factors contribute to the changes in pitch and yaw on our blimp. The engines contribute a torque tending to pitch our blimp, because of the thrust being placed off the center of gravity. The buoyancy, being a force always pointing upward, contributes a restorative force that limits the engine pitching moment - the more buoyant our blimp the less the engines contribute to the pitch.

Another primary source of torques comes from the control surfaces. The rudders provide a yawing moment that is used to rotate the blimp along the Z-axis. The elevators contribute to the pitch by providing a moment at the tail of the blimp.

Drag is a significant but secondary source of torque on our blimp. The drag due to air crashing into the surface of our blimp causes a pitching moment. In a head wind drag contributes to the moment caused by the engines, and in a tail wind drag helps to negate the torque of the engines. Again this is due to the fact that the surface area the blimp presents to the wind is above the center of gravity of our system. Cross-winds (winds perpendicular to the long section of the blimp) are estimated to fall uniformly on the side of the blimp and therefore do not produce a yaw moment.

Other forces acting on our system are virtual mass terms: forces resulting from the fact that the blimp has to displace the air during flight, and gravity. Our blimp is 5 lb heavier than air in order to conform to FAA regulations of a model aircraft.

1.1.1 Blimp Equations

Engine Thrust:

Our 2-horsepower Zenoah G-23 Gasoline Engines each have a three bladed 13-inch propeller with a 10-degree pitch. This configuration with these engines gives us a maximum thrust of approximately 102 ounces per engine. By estimating our thrust control on a linear scale it is possible to determine the specific thrust force produced by the engines. The engine speeds are well matched so roll created by differing propeller RPM can be negated. (The blimp can vector thrust only by rotating the engines along the Y-axis, not by changing the engine throttles independently.)

The engine thrusts in the {X, Y, Z} directions converted to the earth-fixed plane are given by:

Xthrust = magnitude * cos(engine_pitch + pitch) * cos(yaw)
Ythrust = magnitude * cos(engine_pitch + pitch) * sin(yaw)
Zthrust = magnitude * sin(engine_pitch + pitch)

This is given in the Euler-Cauchy matrix

Drag:

Drag is modeled in linear terms. The first step is to take the projection of the current wind vector (which is assumed to be uniform along the entire blimp surface) onto the velocity vector of the blimp. This is achieved by converting the velocity vector into spherical coordinates and replacing the magnitude with the normalized dot product of the wind and velocity vectors. Note that the velocity vector here is not the earth-spaced velocity vector but the blimp-fixed velocity vector.

Once a projected wind vector is available we use a linear drag model. The drag along the blimp-fixed {X, Y, Z} system is found by

Xdrag = area_circle * (wind_velocity-velocity)
Ydrag = area_ellipse * (wind_velocity-velocity)
Zdrag = area_ellipse * (wind_velocity-velocity)

Area_circle and area_ellipse are the cross sectional surface areas of the blimp along the X and Y-axes, respectively.

Control Surfaces:

The blimp control surfaces consist of approximately 1 square meter of rudder surface area and 1 square meter of elevator surface area. In order to model the control surfaces we sum the wind and velocity contributions of the linear drag on the surface. In essence we project the wind onto the control surface's "directional pointing vector" and sum that

with the projection of the velocity onto the "directional pointing vector." This value, multiplied by the surface area of the blimp, produces the elevator and rudder moments.

Positional Error

The blimp's gyroscopes and GPS system provide us with the positional as well as pitch, yaw, and velocity data for our blimp. Since these systems are prone to drift and error random errors that approximate those found in our blimp are introduced into our values. For pitch and yaw this means a +/- 2 degree error is inserted the first minute of flight time and .05 degrees every minute after that. For GPS we add +/- 15 meters every minute to estimate the error. This error is bounded to +/- 15 which is the published accuracy of our GPS system.

1.2 Physical blimp data.

The blimp has two major parts: the envelope and the gondola. The envelope is constructed of an inner PVC bladder with an outer shell of rip-stop nylon. The gondola is constructed of ABS molded plastic with aluminum supports for the engines and servos.

Envelope Mass: 14.651 kg
Envelope length is 9.144 meters.
Envelope diameter is 2.134 meters.

Gondola Mass: 8.233 kg
Gondola length is .853 meters
Gondola width is .254 meters.

1.616 * 2 kg is due to the engines. Gondola mass varies as gasoline is consumed so the gondola mass can vary +/- 1.5 kg.

Estimating the gondola as a square block, since the engine arm lengths are negligible when compared to the envelope, the gondola's moment of inertia is found by:

$$\frac{1}{12} * (.853m^2 + .254m^2) * 8.233kg = .543kgm^2$$

Estimating the envelope as a hollow shelled cylinder we get the moment of inertia being:

$$\frac{1}{2}(14.651kg) * (1.067m)^2 + \frac{1}{12}(14.651kg)(9.144m)^2 = 110.424kgm^2$$

Summing these using the parallel axis theorem we get:

$$I(total) = 110.424 kgm + (14.651kg) * (.940m - .764m)^2 + .543 kgm^2 + (8.233kg)(.764m)^2$$
$$= 116.226 \text{ kgm}^2$$

The blimp flies with approximately 4 to 5 kg of payload.

2. Implementation of the Mathematical Model

The mathematical model of the blimp was implemented in Java. This required programming various vector functionality for cross and dot products as well as transformations from earth-spaced to blimp-spaced coordinates.

Another model built in Matlab was implemented utilizing rudimentary virtual mass calculations, but the details are still incomplete as of this writing. It uses the Runge-Kutta integration method to step through time slices

3. BlimpOS – The Blimp Operating System

3.1 Overview of BlimpOS

The Blimp Operating System is a multitasking, multithreaded operating system written in Java that runs in almost any Java environment. It provides process security and priority levels, scheduling, and message passing services to its hosts. The operating system ensures that processes get adequate time to run as well as preventing errant processes from freezing up mission critical functions. BlimpOS will prevent a rogue program from taking control of the blimp servos, or stealing time away from the navigation system. Currently our BlimpOS runs using the Java JDK 1.3.1 on top of Microsoft Windows 2000 Professional.

Scott Currie – a senior at the University of Pennsylvania and a Blimp Team member – wrote the Original BlimpOS. Because the original OS was written quickly as a proof of concept, the continuation of the project required significant changes to the operating system. These changes will be noted throughout this paper.

3.2 BlimpOS Definitions and blimp.cfg configuration file:

Process – A Java class that extends the class Process. Processes have to implement the boot(), setNewInputStream(), setNewOutputStream() and shutdown() methods. The kernel starts a process by invoking its boot() method.

Properties Manager – A class designed to handle properties passed from blimp.cfg. The properties in question are kernel options, process options, and security options. The first version of the operating system had simple properties that could be passed from the blimp.cfg to the kernel; my subsequent updates now allow processes to have read access to their own configuration information and access to the configuration information of

other trusted processes. Currently we are working on breaking up the configuration files into a dynamic loading system that will allow processes to be inserted and deleted without restarting.

Kernel – The main process that starts all the other processes and runs the scheduler and notifier.

Notifier – This process is called to put the kernel to sleep while another process runs, and then preempts the other process in order to schedule a new one after a set quantum of time. Since each process runs for a set quantum we consider this a cooperative multitasking operating system.

blimp.cfg is the main configuration file for the blimp operating system. This file maintains which processes are to be loaded at runtime as well as preferences and security options for the programs. The blimp.cfg file specifies which processes have communications permission as well as priorities and logging options.

3.3 BlimpOS Kernel and Scheduler

The Kernel and Scheduler are the cores of the operating system. In BlimpOS the Kernel has three major tasks:

1. Start processes as provided in the configuration file.
2. Initiate communications between processes and checking security to determine whether one process has permission to communicate to another. The kernel will create a link between two processes that each process can use to send data to other processes.
3. Forces preemption - Since the Java Scheduler has some significant flaws we needed to write our own process scheduler to fix the problems.

The kernel provides for a system to link two processes together, but the initial version of the OS allowed only one listener and one receiver per process using blocking I/O. The updated version allows for multiple listeners and a non-blocking Queue I/O, which allows a process to post events and other processes to listen for incoming events.

Another function of the kernel is to process the kernel complaint stream tied to each process. Each process can use its request stream to request a communications link with other processes, request more processor time, or even request access to hardware. I am currently working on adding a security model to prevent processes from attempting to access hardware on their own.

The scheduler provides a facility to run through the list of processes and select the next one, based on priority, to run. The system ensures that each process runs and that processes get the priority they need. The priority system was repaired and the system was upgraded to be more robust. The initial version of the operating system could not

handle a process that shut itself down, but now processes can dynamically die and be restarted without having to notify the scheduler.

3.4 Stream Connections

The BlimpOS allows for communications between processes through streams. These streams are either ObjectOutput Streams or of any other stream type that the two processes negotiate prior to compilation. To request a connection stream a process creates a new StreamRequest object, passing a stream title, destination process name, and stream type. The Kernel then checks permissions by looking in blimp.cfg and either calling the setNewOutputStream() of the requestor and setNewInputStream() of the caller or otherwise ignoring the request.

The extension of the process class now contains a specialized event queue. The event queue is created knowing its owner process – since two processes classes cannot have the same name this is allowable. The queue then allows only the calling process to insert elements into the queue, and sends out a pointer to any class that sends out a ListenerRequest to listen to a process. In this fashion we can efficiently implement a method for making a process distribute fresh data. The creating process specifies the queue length and only the newest items are kept in memory. This is useful for GPS data, since if your process stalls for a bit and misses some GPS data events your process won't play catch-up.

Future goals for this aspect of the project are to implement a better device security model and to achieve real time hardware access and performance throttling. The current security model allows any process to access any Java available hardware with only minor configuration file support. This will hopefully be combined into a system where the Kernel locks all devices and can lease them out to other processes, revoking their control as necessary. A function such as this would be useful in cases where a third party wants to test a new control system on the blimp. In this case we would have backups for emergency computer control. Real-time hardware access and performance throttling are necessary to achieving overall computer system stability. Currently our OS has the tendency to starve its host OS making certain processing, like real time video streaming, difficult.

4. Blimp Gondola Modifications and Control Surface Construction

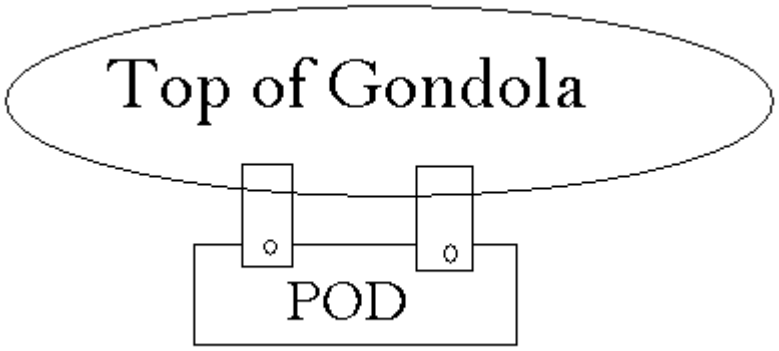
4.1 Introduction and Details

The construction of the blimp's control surfaces happens on a regular basis. We reconstruct the fins in order to repair wood damage from flight stresses and small accidents. Dragging against a floor or ceiling as well as brushing by trees can damage the rudders and elevators. The elevators and rudders are constructed of balsa wood and have an approximate surface area of .7m^2 of elevator; and .7m^2 of rudder.

The gondola is constructed of molded plastic. Because of all of the other supporting hardware - engine servos, gasoline tanks, and radio hardware – its shape and size is inefficient for carrying our extra electronic payload. Therefore we constructed a small electronics package that mounts to the gondola.

4.2 Shielded Electronics Pod

As a result of the high broadband radio noise produced by our blimp engines, especially when they first start, the long cable lengths act as an excellent antenna. This has the horrendous effect of rebooting our computer and can cause our gyroscope readings to skew. (More details are provided in section 6.) In order to shield the laptop and cables as well as provide a mounting space for our electronics, we created a miniature electronics pod. Approximately half the size of the gondola it mounts underneath, the miniature pod is attached through dowel rods that slide into retainer holes and are secured with friction and cotter pins.



The miniature pod is constructed of a Tupperware container approximately 16 inches long by 9 inches wide by 6 inches deep. Most of the container is drilled away, leaving a cross connected rib and an outer rim. For strength we shrink-wrapped the pod in Mylar, which provides a relatively strong body that is electrically conductive. Mylar's poor conductivity makes it a poor RF shield. In order to aid its shielding capabilities we put on many layers of film and wrapped some outer layers in thin copper wire. The pod connects to the engines as a ground to assist in shielding out engine RF noise. Balsa wood reinforcements are placed on key stress points in the pod.

Thanks to the light construction the pod weighs only ½ lb. Its RF shielding capabilities are rather small but good enough for protecting the laptop, which wouldn't operate prior to shielding. Another step we took to protect our electronics was to replace the spark plugs with resistive spark plugs. Resistive spark plugs have a resistor in parallel with the plug, reducing RF emissions. Luckily our shielding only slightly attenuates wireless Ethernet. (802.11b).

Future tasks for this pod involve using a shielding cloth instead of Mylar. Besides being a better shielding material it could be significantly stronger and conduct heat better.

Mylar contains most of the heat inside of the pod, quickly heating all of our electronics. Inclusion of a small 12-volt fan may help this situation.

5. Feedback Sensors

5.1 GPS System

The GPS system is a Garmin GPS36. The GPS36 can take any input voltage from +6 to +30 V and has two serial inputs and one serial output. The inputs are used to program the device via NMEA 0183 ASCII Interface Specification and to provide differential correction beacon interfacing with an RTCM Recommended Standard for Differential Navstar GPS Service, Version 2.0, RTCM Special Committee No 104.

The outputs are NMEA 0183 Version 2.0 Standard Outputs with a few Garmin extensions. The exact specification is listed on Garmin's website: http://www.garmin.com.

The GPS currently runs off a 9V battery, which because of the low energy density of the batteries gets us approximately 1 hour of battery life. We are planning to introduce an onboard 12V electronics bus to power all the electronics and supplement the laptop's on-board power.

The GPS is programmed via a Garmin utility prior to runtime to select the specific GPS sentences we would like outputted. The sensor is then hooked into the flight laptop, and the operating system GPS module is started. The GPS module reads the GPS and processes the data, redistributing it to any other process that wishes to listen.

Mounting of the GPS is done via a patch of Velcro at the top of the blimp envelope over the center of mass. Despite its distance from the center of mass this is the most functional place to mount the GPS so that it has a clear view of satellites.

The GPS accuracy is approximately +/- 5 meters.

5.2 Gyroscope

We have gyroscopes on the blimp interfacing via RS232 to measure the pitch and yaw of the blimp. The mounting of the gyroscopes is crucial to their performance as drift can be very noticeable in such a system. The gyroscope drift is slightly corrected for in the system but vibrations still cause errors. In order to tie the gyroscope as closely to the blimp motion as possible, the bulky electronics have been segregated from the actual sensor heads, and the sensors themselves are mounted directly to the envelope. Wire length is an issue since any small noise could disrupt our gyroscope reading. Therefore the electronics are also mounted on the envelope just a few centimeters away from the reading heads. This detached method provides for gyroscope heads that do not sway or wobble as much as keeping them tied to the bulky electronics would.

A student working in the University of Pennsylvania GRASP lab created the gyroscope reading hardware. It's called the DAX and its documentation is included as Appendix 1.

Problems we are encountering with our DAX are attributed to drift of the gyroscope due to engine and wind vibration. We have solved some of this by creating a strong mounting for the two gyroscope heads and by increasing the airship's turgidity.

5.3 Radio Controller to Computer Interface

Using the RealFlight RC Transmitter Interface-™ we have developed a system that transfers RC controls to the parallel port in order to record how specific control inputs correlate to changes in the blimp's dynamics.

6. Numerical Models Through Data Acquisition

Our plans for creating the control system involve first getting some sample control data. We have developed a system where we can monitor the radio controller outputs to the servo and combine their data with the GPS and Gyroscope data. Combined, all three pieces of information will give us an experimental model for how the blimp operates. Our first scheduled set of data collection is scheduled for the end of August.

In order to correlate this data we will first walk the blimp to certain checkpoints, and then fly it along a predetermined path to hit those checkpoints. Once each checkpoint is targeted we can evaluate a numerical model and try and to correct for any steady state noise. (I.e. Wind.)

7. Non-Technical Details

The Blimp Project has required a great deal of non-technical, behind-the-scenes work in order to keep it alive. Tasks I have had to deal with over this summer included repair of previous blimp damage that required professional tailoring; locating and securing funds to continue our insurance policy; fundraising; and organizing manpower to hold flight tests. This project required a small business management attitude – something that occasionally held research in limbo.

This summer I have secured some more funding for our project as well as a full year of insurance. I have organized trips to Carnegie Mellon University for flight-testing as well as oversaw the repair and maintenance of the blimp.

8. Why the blimp is not electric.

The blimp runs on its standard Gasoline engines because going to electric engines would require huge amounts of batteries that would last us only 9 minutes of flight time. The large power output required of the blimp as well as the fact that we need positive thrust in order to just hover, means that electric engines equivalent to our gasoline engines would have to provide approximately 1400 Kilowatts each! This is impractical and until we can

get higher energy densities (i.e. fuel cells) onboard an electric blimp of this size is not likely.

We have explored many options – i.e. powering from the ground, etc. Unfortunately powering the blimp from the ground would not only severely limit its mobility but also blow out the motor speed controller due to wire inductance.

9. Discussion and Conclusions

All in all the Blimp Project was a successful attempt at modeling the dynamics and integrating a sensors package into our model airship. We have successfully created multiple models for use in mathematical simulations, mounted sensors to the blimp, created radio noise shielding, redesigned and redeveloped an operating system for the operation of the blimp, developed software for interfacing our sensors and radio control transmitter into the operating system, and managed the project's business details to keep it funded and insured.

The completion of the feedback laws and automation system is an ongoing project that will continue through and beyond the writing of this paper. Very soon we will have started to prototype a control system designed to stabilize the blimp and pilot it through 3-space. Further refinement of the mathematical model will happen as part of this project in September, and we will have more information on our numerical model by then also.

10. Recommendations

This is an ongoing project that is still continuing past the sponsored SUNFEST NSF-REU time. Being a University of Pennsylvania student I have the privilege of continuing my work throughout the year. I can recommend only that work continue on the refinement of the mathematical model and feedback loops as a higher priority now that the sensor mounting is completed and integrated.

11. Acknowledgements

I would like to thank the NSF for their support of the REU program and SUNFEST. I would also like to thank Microsoft for sponsoring me as a Microsoft Fellow. Without their support this project would have been impossible! Dr. James Ostrowski and Dr. Jorge Santiago have been great supporters of this project from its onset, as well as Dr. Eduardo Glandt, dean of the School of Engineering and Applied Sciences. The entire Science and Technology Wing Blimp Team, especially Benjamin Kin Tang, Mike Bruni, Dan Levin, and Chris Hoess for their vision and perseverance throughout.

In conclusion I would also like to thank Dr. Jan Van der Spiegel for making SUNFEST possible.

12. References

1. Bueno, S.S; Elfes, A; Bergerman, M; Ramos, J.G, AURORA: A Semi-Autonomous Robotic Airship for Environment Monitoring, Technical Report LRV-1997-11, Automation Institute / CTI, 1997.

2. Etkin, Bernard and Lloyd Duff Reid, Dynamics of Flight 3rd edition, John Wiley and Sons, New York, 1996.

3. Varella Gomes, Sergio B. and Josue Jr. G Ramos, Airship Dynamic Modeling for Autonomous Operation, Proceedings of the 1998 IEEE International Conference on Robotics and Automation, Leuven, Belgium, 1998, pp 3462-3467