Design and Implementation of a Wirelessly Controlled Blimp

By Donghyo Min, Eric Hare, and Josh Peterson

Table of Contents

Introduction

For our final project in CSE 466, our group constructed a remote-controlled wireless blimp. We developed a PC Graphical User Interface to control the blimp. Using Texas Instruments microcontrollers, we constructed a UART relay for wireless transmission of commands and other data. On the blimp itself, a camera is mounted to allow for video recording of the flights.

This paper presents an overview of related works, a description of the system design and implementation from the firmware and hardware perspectives, and analyzes the performance of key aspects of the blimp.

Related Work

Flying vehicles have been used elsewhere as a teaching tool. At the University of Washington, Dr. Shwetak Patel's course on Embedded Microcomputer Systems (EE 472) [1] has students program a controller for the Parrot Drone[2] remote-control (RC) helicopter, and students have their systems compete on an obstacle course for the final project. Also at UW, a student team also constructed their own RC blimp in 2010[3].

Small blimps are found in a variety of interesting contexts. They are marketed as toys, and our envelope is similar to many of these[4]. They even been used as a means to fly speakers in a sound artwork[5]. Though only tangentially related, weather balloons often use embedded systems, such as a group of MIT students' $150 system which was able to reach almost to the edge of space[6].

Our system uses the TI MSP430 microcontroller, which has also been used in other flying vehicles such as a quadcopters[7]. We also employ a small camera for recording in-flight video, the 808 keychain camera[8], which is popular in hobbyist applications.

System Design and Implementation

Hardware

Our blimp system uses two eZ430-RF2500 wireless chips from Texas Instruments to handle communication between the PC and the blimp[9]. The eZ430 uses an MSP430F2274 microcontroller. These chips are statically linked when powered, eliminating the possibility of an accidental wireless connection to some other chip. One eZ430-RF2500 is connected by a USB cable to the computer, while the other is connected directly onto the blimp board itself.

The blimp uses an MSP4305510 microcontroller from Texas Instruments[10]. The various blimp functions are connected to output pins of this controller. Specifically, the motors are connected to P1.0 - P1.5, and the infrared sensors uses a combination of output pins and circuit logic to select the appropriate transmit source.

Firmware

Perhaps the most significant aspect to our firmware is the UART relay developed to send commands from the PC and receive a reply from the blimp. First, the USCI peripheral on the eZ430 was configured to allow communication with the PC. We chose to encode commands

send from the PC as opcodes. The table of commands used is listed below:

Command	Function	Description
0	Stop	Stops and disables blimp motor and IR controls until a “Run” command is received
1	Run	Activates blimp motor and IR controls, sets output pins high (standby mode)
2-18	Left Fan Speed	Set the speed of the left fan, ranging from 2 (highest reverse speed) to 18 (highest forward speed). A 10 turns off the fan motor
19-35	Right Fan Speed	Set the speed of the right fan, ranging from 19 (highest reverse speed) to 35 (highest forward speed). A 27 turns off the fan motor
36-52	Altitude Fan Speed	Set the speed of the altitude fan, ranging from 36 (highest downward speed) to 52 (highest upward speed). A 44 turns off the fan motor
53-57	IR Sensor Data	Returns accumulated sensor data from the blimp to the PC. 53 = Forward IR Sensor 54 = Reverse IR Sensor 55 = Left IR Sensor 56 = Right IR Sensor 57 = Down IR Sensor
58	*Power Data*	*Returns the current voltage of the MSP supply voltage on the blimp* **(Not currently implemented)**

Table 1: List of Command Supported by Blimp

Because we encoded commands as single bytes (opcodes), we were limited to 256 possible unique commands. However, the UART on the eZ430 implements a command buffer, storing up to 20 different commands and executing them in a control loop. This allows more complex maneuvers to be performed by the blimp, such as quickly turning the blimp in a particular direction.

As per the commands table above, our fan motors allow for several speed configurations. Each fan motor is capable of operating at 8 forward and 8 reverse speeds, and independently of one another. This was accomplished using a hardware PWM on the 5510. A timer is initialized to a value ‘PERIOD’ such that the motors operate at about 200Hz. The other timers are then set such that an appropriate duty cycle is achieved. Whent the blimp is initially run, all the timers are set to this ‘PERIOD’ as well. Therefore, to set the left fan motor to the maximum speed, the value of the right fan motor’s PWM need only be set to 0. This clean implementation allows for extensibility in the hardware design, by (for instance) having even more fine-grained speeds or limiting the speeds as necessary. All such values are defined as constants in the 5510 code to allow for these modifications.

Although commands from the PC are encoded as single bytes, replies from the blimp can be arbitrarily long strings. We accomplished this by using a reserved character to denote the beginning and the end of a blimp reply. This allowed us to elegantly implement the IR sensor functionality. The IR sensor desired is 'selected' by toggling certain output pins. The reading from this sensor is then summed 40 times and filtered. The accumulated value is send from the blimp's UART to the eZ430 on the blimp board, which then transmits the data back to the PC. The full communication cycle appears in the figure below:

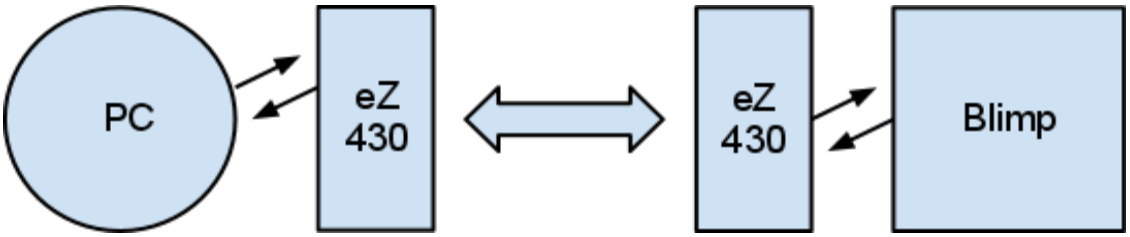

Figure 1: Communication link between PC and Blimp

Graphical User Interface

Our Graphical User Interface (GUI) was written in the Java programming language. It uses the Swing library, along with a third party RX/TX module[11].

Figure 2: Java GUI for Blimp Control

The GUI allows the user to quickly connect to a communication port on the PC, in order to transmit data to the eZ430. Beneath this are buttons for the stop and run commands, followed by buttons for motor control. The GUI has hard-coded buttons to adjust the blimp to turn left, right, go forward, and go backwards. When pressed, these buttons rapidly send a sequence of commands to the blimp via UART. The interface is event-driven in that the main control loop has very little functionality. Instead, events are fired upon button presses. This behavior is similar to that of an operating system event manager, using a queue to determine which commands to execute first.

Each button is also connected to a particular key on the keyboard. This allows the GUI to be controlled by external input devices, such as a game controller. Furthermore, the GUI contains IR sensor capability, including graphical display of the results. Finally, the GUI adds an automatic altitude feature which, when activated, will monitor the downward-facing sensor and adjust the altitude of the blimp if it detects that the blimp is nearing floor. It accomplishes this by repeatedly polling the sensor in between execution of other commands. If a particular threshold value is achieved by the sensor, then the top motor is automatically activated at a high speed to provide upward thrust. When an exit threshold is reached, the motor deactivates. Our GUI supports this functionality while still allowing the user to send commands through the interface, so operation of the blimp can continue.

Extra Features

We implemented a few extra features beyond the requirements of the lab. They are listed below.

1. Collision Detection Framework
We modularized the algorithm for altitude control so that it works with all of the sensors. This requires two changes to the GUI code. First, the constant for the sensor to use must be set from IR5 to any of IR1 - IR5. Second, the threshold value (the value at which the blimp is to avoid collisions) must be modified. Given these two changes, the blimp will attempt to avoid collisions in all directions. Note that as a drawback, only one sensor may be used for collision detection at a time.

2. **IR Sensor Graphs**
These are described briefly in the GUI description above. In addition to simply graphing the data, the GUI will display up to 5 old results as black lines on the graph, so that a history of the readings is present. An “IR All” button exists on the GUI to poll all 5 sensors quickly and report the results to this graph, as well as the console.

3. *(Unfinished)* **Power Management**
We began implementing power management, to detect whether the MSP supply voltage falls below a given threshold. The command is present in the command list, and the function in the blimp firmware has commented code that attempts to achieve this.

Results

Overall, we considered our project to be a great success. Our blimp is fast, responsive, and the wireless communication protocol was able to both transmit and receive accurate data quickly. When using Realterm to transmit IR commands and receive the results back to the terminal, we were able to achieve a**n average data rate of 9 samples received per second.** Other commands were even faster, as they did not necessitate a reply back to the PC. We felt that the fine-grained motor control was a big plus for our project, as it allows significant flexibility in how best to operate the blimp. Nonetheless, there were aspects of the project that could use improvement.

1. **Automatic Altitude Control**

An unfortunate discovery of our testing was that two IR sensors in particular seemed weaker than the other three. We quantified this by taking 10 accumulated samples at various distances from the sensor and averaging them. (Note that some truncation takes place in the code and so the summed values are in increments of 20) The results can be seen below.

Sensor	**Distance From Hand**	**Value**
Forward	12 Inches 6 Inches 3 inches	34,080 26,400 10,460
Reverse	12 Inches 6 Inches 3 Inches	37,820 36,740 37,020
Left	12 Inches 6 Inches 3 Inches	33,700 23,700 11,980
Right	12 Inches 6 Inches 3 Inches	35,000 26,560 19,540
Down	12 Inches 6 Inches 3 Inches	36,600 35,340 27,280

Table 2: Performance over Distance of IR Sensors

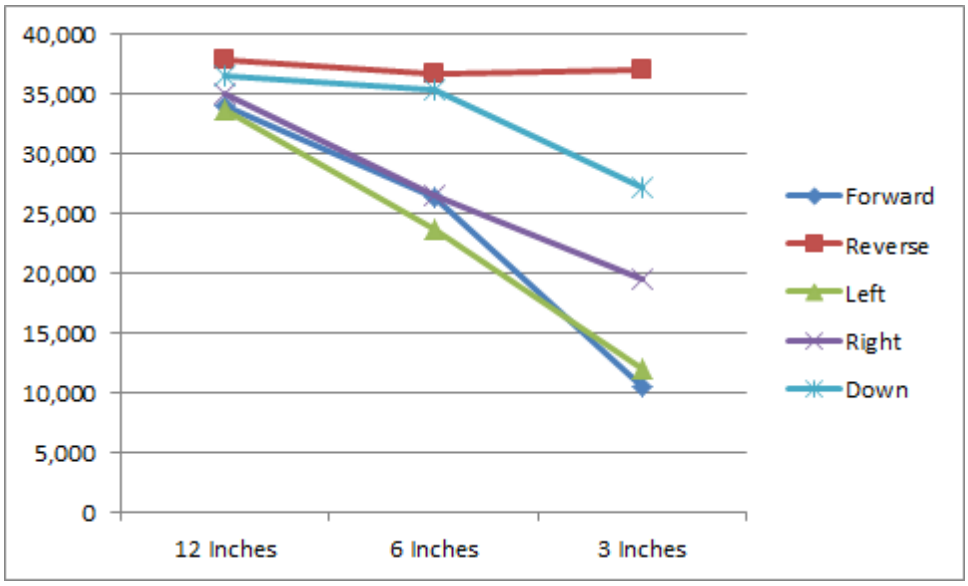

Figure 3: Graph of performance over Distance of IR Sensors

As you can see, the resolution of our down sensor is fairly weak. We were able to calibrate the altitude control to reliably activate the motor when within a few inches from the ground. However, when the blimp is descending too quickly, this does not give it enough time to reverse the momentum. Furthermore, because the threshold value is so borderline, the blimp will occasionally activate the motor when well above the ground. We are not certain of why our down sensor appeared so weak, but perhaps some further amplification of the sensor readings could have helped solve this problem.

We were hoping to characterize the sensor data further by abruptly changing the threshold value and observing how the blimp responds. However, because the range of our altitude detector was minimal, this proved nearly impossible. Instead, we caluclated a measure of the noise of each sensor when held in a steady position with even room lighting. We did this by sampling 10 values from each sensor and calculating the standard deviation. The results are as follows:

Sensor	**Standard Deviation**
Forward	24.51
Reverse	9.65
Left	37.12
Right	34.95
Down	45.44

Table 3: Standard Deviation of IR Sensor Measurements

Note that the forward and reverse sensors had statistically significantly less variation than the

others. The reverse sensor was particularly good about this, although as can be seen from the graph of values over distance, it did not perform well from the standpoint of detecting infrared light. The down sensor had the most variation as suspected, although not by a statistically significant margin.

2. **Motor Control Using a Video Game Controller**
Our GUI was designed without the controller use case in mind. A consequence of this is that there is a very steep learning curve when using the controller to fly the blimp. Since we designed our GUI with hard-coded buttons for toggling all fans at once to turn certain directions, mapping joysticks to such commands proved cumbersome. We discovered that a way to alleviate this problem is to activate the command for a short time, and then quickly revert the motors to idle (similar to how a joystick reverts to the center position). However, we did not have time to adjust our GUI to support this use case for the demo.

3. **Java Graphics Library in GUI**
For reasons that are not entirely clear, our GUI caused numerous problems when attempting to activate altitude control mode. Evidently, multiple threads would attempt to access the UART at the same time. At some indeterminate point in time in the future, the threads would deadlock and the algorithm would freeze. Although we were able to avoid part of this problem by using a mutex, it turns out that multiple threads would run the main function as well. We had to implement some less-than-ideal changes to the code in order to avoid this situation, and a consequence of the result is slower readings from the IR sensor. (About two per second, on average). This may have also negatively impacted the performance of our automatic altitude control mode.

Future Work

Our system was constructed in such a way that it would be easily extensible to do object avoidance using the front, rear, left, and right IR sensors (see the extra features section). This could potentially lead to a blimp that is able to wander around a space without hitting objects, or could be used as a fail-safe against human error when the blimp is piloted by people.

We began to prepare to use 3d mouse: 3d connexion Space Navigator[12], which would employ the Java Native Library JNL with Windows built in raw data support for Human-Interface Devices (HID)[13]. This would be an interesting and possibly intuitive way to navigate the blimp.

As mentioned in the extra features section, having a battery monitor would be a useful feature for avoiding battery damage and could be implemented using the MSPF5510 Supply Voltage Monitor.

References

1. http://abstract.cs.washington.edu/~shwetak/classes/ee472/
2. http://ardrone.parrot.com/parrot-ar-drone/usa/
3. http://www.washington.edu/news/archive/id/58446
4. http://www.rctoys.com/rc-products/MACH-3Z-GOODYEAR.html
5. http://www.youtube.com/watch?v=dnNI297XV1I
6. http://space.1337arts.com/
7. http://www.43oh.com/2010/11/msp430-based-quadcopter-ez430-rf2500/
8. http://www.chucklohr.com/808/
9. http://www.ti.com/tool/ez430-rf2500
10. http://www.ti.com/product/msp430f5510
11. http://rxtx.qbang.org/wiki/index.php/FAQ
12. http://www.3dconnexion.com/products/spacenavigator.html
13. http://www.blog.kslemb.com/doku.php/en/projects/globx/java_hid