

Hugh Dillon | Zac Hawkins | Rick Holloway | Ben Smith | Hunter Thorington

MAILBIRD

**CYCLE 1 REPORT**  
Wednesday | March 5, 2014

Contents

[1 Project Description 2](#_Toc378085694)

[Abstract 2](#_Toc378085695)

[Executive summary 2](#_Toc378085696)

[Project Definition 2](#_Toc378085697)

[2 Technical Approach 3](#_Toc378085698)

[Design Considerations 3](#_Toc378085699)

[Potential Problems 4](#_Toc378085700)

[Coding limitations 4](#_Toc378085701)

[Infrared White-out 4](#_Toc378085702)

[Carrying Capacity 5](#_Toc378085703)

[New Buildings 5](#_Toc378085704)

[Landing Module 5](#_Toc378085705)

[Market Constraints 5](#_Toc378085706)

[Design Standards 6](#_Toc378085707)

[3 Management Approach 7](#_Toc378085708)

[4 Budget 7](#_Toc378085709)

[5 Timeline 8](#_Toc378085710)

[6 Facilities To Be Used 9](#_Toc378085711)

[Design Lab – Hardware construction 9](#_Toc378085712)

[Labs 308 & 310 9](#_Toc378085713)

[Testing 9](#_Toc378085714)

[7 Disposition Agreement 9](#_Toc378085715)

[Appendix I 10](#_Toc378085716)

# 1 Project Description

## Abstract

This document details the preliminary approach to an autonomous package delivery system. A summary is given of the design specifications, and of the system that will be created to satisfy those conditions. Consideration is given to design decisions, technical specifics for implementation, and the anticipated use of the completed system. In addition, procedures are defined for project management, financial operations, and distribution of equipment after project completion.

## Executive summary

MailBird is a proof-of-concept automated mail system. A successful guidance system will use GPS to bring a quadcopter within five feet of the desired landing area. Subsequent to GPS localization, it will use a custom built guidance module to dock within a tolerance of 1 inch. The technical aspect of the project will be designing an aircraft guidance module and associated ground station (if necessary) to land in a precise location carrying a deliverable. This will be accomplished with a quadcopter equipped with an augmented precision external landing module (APELM), for extreme accuracy. The APELM will be designed to use infrared optics and LEDs to determine position. This module will interface with the flight controller to guide the docking procedure. After creating an effective, precision landing system, if time allows the system will be extended to include a software suite and mail delivery peripherals. The final result will be a comprehensive autonomous delivery system supported by a suite of a custom-designed scheduling, pick-up and delivery software.

## Project Definition

The MailBird project arose out of an Auburn University electrical engineering senior design section. After discussing numerous ideas, the design team decided that a satisfying project would involve a quadcopter. The intention was to use the unique flight capabilities of a quadcopter to accomplish a common task from both a literally and figuratively different perspective. Brainstorming with this constraint resulted in a large amount of ideas. One potential application placed a security camera in a difficult to reach area, such as the roof or exterior of a building. Another idea gave the quadcopter the ability to determine its surroundings and avoid obstacles while traversing from one location to another. Ultimately, elements of each idea were combined into a system that could carry an object and determine the location of a landing station in an environment. A parcel delivery system was the realization of the final idea; it would accomplish a common task from a different perspective, and it involved locating a delivery location for the parcel it would be carrying.

# 2 Technical Approach

## Design Considerations

We considered four options for transporting a package from one location to another via the quadcopter. The first option was solely using GPS to deliver the package to a predetermined location. However, the GPS location varies within a five foot radius. Since the required tolerance is within an inch of the desired landing area, GPS only will not suffice. The following three options all use GPS to bring the package within a five foot radius and use the custom guidance module to land.

The following three options were considered to precisely land the quadcopter: a camera to locate infrared LEDs on the ground, an optical camera to locate a red square on the ground, or an ultrasonic device to detect the frequency of the reception of sound waves. The following Pugh chart compares the three options.

Table 1 – Landing Implementation Comparison



The Pugh chart above is weighted to emphasize fulfilling the goals and enabling reproduction of Mailbird. All of the above options are complex in design and precise when it comes to landing so there is no differentiation in these two categories. Because of the affordability and reliability of the infrared camera and LEDs the net score is higher for option 1 (IR LED), which is the design chosen for Mailbird.

Another design consideration that was analyzed by Pugh charts was whether to fly from dock to delivery location on a predetermined path or by going to the location with collision avoidance implemented on the quadcopter.



Table 2 – Guidance Implementation Comparison

The Pugh chart above is weighted according to the difficulty of the two choices and in order to not break any regulations already established by the FAA. Although collision avoidance in theory sounds good the practicality and difficulty of it compared to a predetermined route makes it a bad option.

The last Pugh Chart is shown to demonstrate why the design chosen is for general quadcopters and not just specific to the quadcopter used in MailBird. With a ratio of quadcopter size to landing pad size the design will hopefully be able to be implemented on any ArduCopter device.

Table 3 – Compatibility Comparison



## Implementation

### Flight mode control

Once data is received from the camera regarding the position of the quadcopter relative to the infrared LEDs, it is integrated into the controller so that the quadcopter hovers directly over the landing pad (and hence the LEDs). This integration was accomplished mainly via two parts of the Arducopter code—the user code file provided to integrate custom code with the Arducopter and the preprogrammed commands to “loiter” in a constant position.

The LOITER flight mode enables the quadcopter to hover over a constant position by maintaining a constant GPS location. When the flight controller switches into LOITER mode, the quadcopter picks its current GPS position as the position it wants to maintain—this is the target position. It continually calculates its error, or deviation from that position, by continually checking its current position and comparing that to its target position.

Once the controller calculates its deviation from the desired GPS position, it tries to adjust using a software defined PID control loop. Via the control loop, it calculates the desired pitch and desired roll of the quadcopter to adjust its position as required. In test flights, LOITER mode has worked very accurately—the quadcopter holds a steady position very well.

A separate flight mode (which will be referred to as IR\_LAND mode) is used to land the quadcopter directly on top of the infrared LED landing pad. This flight mode is integrated with the quadcopter via the user code file, which contains several loops that are run at different frequencies (50 Hz, 10 Hz, etc.) These loops are used to run the camera and control the quadcopter when it is in IR\_LAND mode.

The flight controller updates its pitch and roll when it is IR\_LAND mode using the same functions that it uses in LOITER mode. The camera can calculate its distance from the center of the LEDs using the triangulation algorithm, which is discussed in another section of this report. This distance, calculated in cm, is the error, or deviation from the desired position. This error is passed to the same PID control loop that is used to adjust position when the quadcopter is in LOITER mode. The loop then returns the values for desired roll and desired pitch to adjust the position of the quadcopter so that it is hovering directly above the LEDs. The control loop is called from the 10 Hz loop in the user code. A flag is used to tell the control loop to look for error from the camera and not the GPS.

### MAVlink communication

The goal of our project is to have an autonomous system that can be easily programmed to make deliveries to predetermined locations. This means that a GPS track will be assigned autonomously, and the additional APELM module must be able to interface into the autonomous program. In order to put the quadcopter into a flight mode that is not included with the APM: 2.0 Mission Planner software, we went to the underlying communication protocol that the Mission Planner GUI is built on, and then updated the internal code to react to a set-flight-mode packet containing our new infrared landing flight mode. The APM: 2.0 Mission Planner has a method of setting the mission waypoints to a raw numerical value instead of a default command (with a description pre-programmed in the GUI). About 40 of these parameters are unused, but can be used with an existing MAVlink command and simply added as a parameter. Therefore, we added a flight mode to the MAVLink enumeration in the flight controller code (the ArduPilot itself) so we could set the flight mode when a new packet comes into the controller carrying a previously unrecognized parameter. Whenever the MAVLink receiver in the flight controller receives a MAV\_CMD\_DO\_SET\_MODE command with an argument of 14, it knows to trigger the infrared landing procedures.

However, we also had to be able to test the quadcopter in a controlled environment by probing its response to certain behaviors. MAVLink was the primary method we used to determine if implementations in the flight controller code we were working correctly. By commands via a python library called pymavlink, we were able to repeatedly test switching flight modes under different test scenarios. We were able to gather data on what behavior to expect whether we had GPS lock or not, whether or not the motors were armed, or if our I2C bus was getting flooded. By issuing a heartbeat request on the USB interface, we could glean information about the actual state of the quadcopter code without actually flying it. This is was very useful as testing a new flight mode is a significant, costly undertaking otherwise.

Note that we did not create a new MAVLink command to interface with the flight controller, rather just added a parameter to an existing MAVLink command. With the addition of pymavlink, we were able to test and verify correct operation of various features without having to take apart the inner workings of a protocol which is very complicated and versatile.

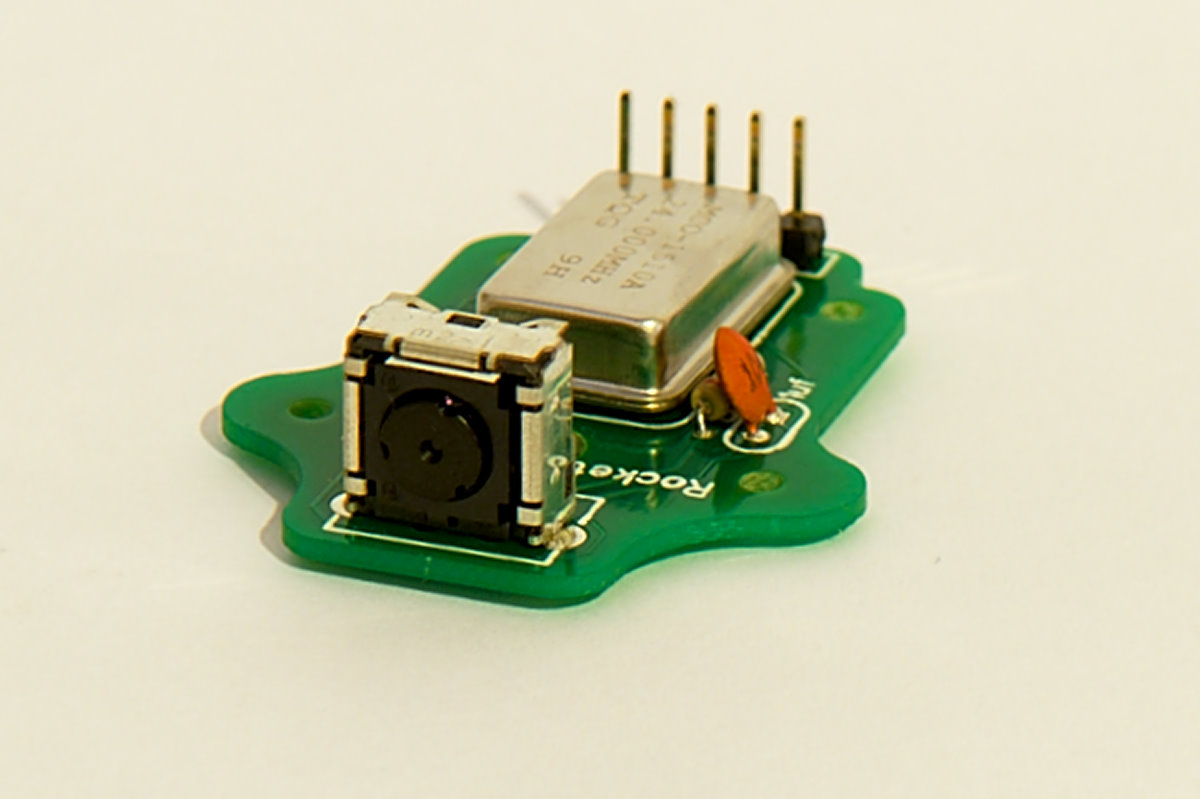
Further, this gives us an alternative to the APM:2.0 Mission Planner software to use when debugging. Note that this will require purchase of a telemetry module, which we currently do not have.

### WiiCamera Library

To enable the WiiCamera to work properly on any type of Arduino board a library was written to streamline initialization and reading from the camera. The Library is provided in the form of two files. WiiCamera.cpp and WiiCamera.h These two files define the object known as WiiCamera (or ircam in the APM code). The library allows for code reuse and simplified user code in the APM flight module. The library can be found in Appendix IV.

Integrated WiiCamera Circuit Board

During Cycle 1, it came to the group’s attention that the large scale of the prototype camera circuit was too fragile and cumbersome to used on the quad copter. A pre-made circuit board was found online at <http://rocketbrandstudios.com/> this integrated circuit board had traces for all the necessary circuity and a sturdy 5-pin connector. This board was ordered and assembled using surface mount components similar to the original circuit. The advantage to this board over the prototype was that is was sturdy, had no external wires and had mounting holes so it could be attached to the quad copter. The new completed WiiCamera board is shown below.

WiiCamera Circuit Board

### Triangulation algorithm

The triangulation algorithm allows the quad copter to determine its position over a landing pad with IR LEDs imbedded. The algorithm takes a set of two X,Y positions as inputs and outputs the quad copters x,y offset and elevation. The algorithm runs on the APM board in a 100Hz loop. The algorithm’s entire code can be found in Appendix IV.

Algorithm Constants

The algorithm’s constants are determined from data gather from the Wii Camera Data sheet found at: <http://wiibrew.org/wiki/Wiimote>. These constants are used in many calculations and are camera specific. They are detailed below.

#define X\_PIX 1024 // number of pixels in X dimension of camera

#define Y\_PIX 768 // number of pixels in Y dimension of camera

#define X\_CENTRE X\_PIX/2 // X coordinate of Centre Pixel in Camera

#define Y\_CENTRE Y\_PIX/2 // Y coordinate of Centre Pixel in Camera

#define X\_FOV 47 // Field of view of Camera across X dimension

#define Y\_FOV 35 // Field of view of Camera across Y dimension #define PIX2DEG 0.045 //(X\_FOV / X\_PIX) # of pixels per deg of view

#define TARGET\_MAX\_WIDTH 200 //width of IR blobs on IR target in mm

Algorithm Inputs

The algorithm takes two inputs. These two inputs are Blob structs as defined in the camera code. The structs contain the x and y pixel position of each blob being tracked.

// Structure to hold blob data

struct Blobs

{

int X;

int Y;

byte number;

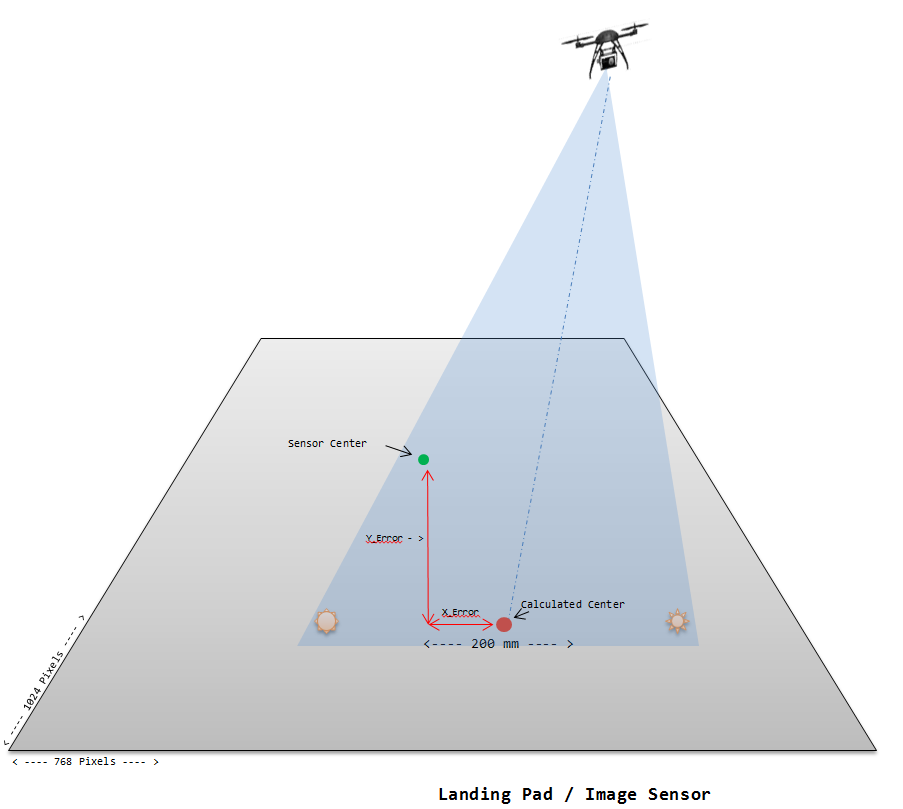
};

Algorithm Outputs

The algorithm output is the calculated location of the quad copter above the two LED blobs. This output is in terms of X and Y offset from the center of the blobs. These values are fed into the APM PID controller as X and Y position error. The PID uses these values to correct its position accordingly.

Algorithm Operational Flow

Algorithm Geometry



### Changes in IR camera integration

The design decision was made to use an IR camera to accurately land the quadcopter. In order to implement this module successfully into the Mailbird system the camera needs to work cohesively with a compass, GPS, and other components attempting to communicate with the control in order to maintain stable flight. Design considerations for how to combine the IR camera sensor with the flight controller included amount of additional external hardware, work/time required for full integration, and information collision avoidance. Table 4 illustrates how the final design decision was made, with emphasis on limiting the amount of external hardware due to inherent weight and spatial limitations.

Table 4 – Communication Comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Weight | I2C | SPI | Analog Input |
| External Hardware | 2 | + | - | - |
| Work/Time Cost | 1 | - | + | - |
| Communication Collisions | 3 | + | ? | - |
| + |  | 5 | 1 |  |
| - |  | 1 | 2 | 6 |
| Net Score |  | 4 | -1 | -6 |

Because of this comparison of alternatives and the fact that we still had ample time to give to solving the communication problem the MailBird design uses and I2C interface with the IR camera.

Another important note is the properties related to I2C the important ones in this case being the ability to access multiple devices on the same line and the requirement of acknowledgement bits. Since the flight controller already operated with inputs from a compass and GPS, both using the I2C line, there was already a hardware scheduler in place to insure communication conflicts will not occur. This scheduler was also easily tapped into for use the with the IR camera module.

Problems occurred when attempting initial communication with the camera. This was due to I2C registers not being set by the flight controller. The problem was resolved by initializing the five necessary registers, interrupt pending, enable acknowledgements, line enable, interrupt enable, and line start. Creating a separate function called by the IR camera initialization routine to initialize these registers caused the camera to function properly. Correctly functioning I2C protocol and a scheduler allowing for limited communication conflict the camera module is currently able to gather data about IR blobs and successfully transfer this data to the triangulation algorithm.

## Potential Problems

The following is a list of potential problems we could run into with implementing the IR camera and predetermined path onto the MailBird:

### Coding limitations

Our group as a whole is not familiar yet with Arduino programming and how to respond to the infrared camera on the ArduCopter.

### Infrared White-out

How will the sun affect the IR camera? Will the sun reflect off the ground and cause the camera to see nothing but IR light? If this is the case, one alternative we have considered is building a focal point made out of PVC pipe painted black to block out some of the other light shining into the camera.

### Carrying Capacity

What is the maximum size package that the Arduino can carry? What is the ratio of increased package weight to battery life? How much power is the camera circuit going to take from the battery?

### New Buildings

When construction is going on the MailBird will continually have to be reprogrammed due to changes in geography and obstacles like cranes.

### Landing Module

How much is it going to cost each customer to buy/build the landing zone?

### IR Camera Lens Modification

Ordered wider lens for the IR camera how will the implementation work? Will it increase drastically or will we still need more intensity and a wider viewing angle for the LEDs? Does attempting to compensate for wind push the Mailbird out of field of view of LEDs?

### Simulations

How can we test without taking off an arm, burning a motor, or destroying the quadcopter and/or room?

### Weather

Can we compensate for bad weather? If so, how?

## Market Constraints

The MailBird system is intended to deliver small packages from a central hub to predetermined mail-drop locations. The idea is directed towards a market that requires frequent or scheduled delivery of packages with essentially immediate delivery. Market attributes including economic feasibility, manufacturability, public health and safety, social opinion, environmental impact, and political implications were taken into consideration when designing the MailBird system.

In order for the project to be marketable at all it must first and foremost be designed with the idea of profitability. With an low investment cost for the entire system and even less when equipping the tracking module to an already built Arduino based delivery system the MailBird is easily and cheaply integrated into any delivery role. However the true gain comes from savings in labor and fuel costs. The MailBird is completely electric requiring only to be charged after every fifteen minutes of flight time. LiPo batteries power the system mainly due to their power to weight ratio, but also because they hold long battery life up to 1000 charge cycles and a cheap replacement cost. With a designed ten minute flight radius and instantaneous delivery the MailBird is designed to increase the productivity and satisfaction of an entire campus.

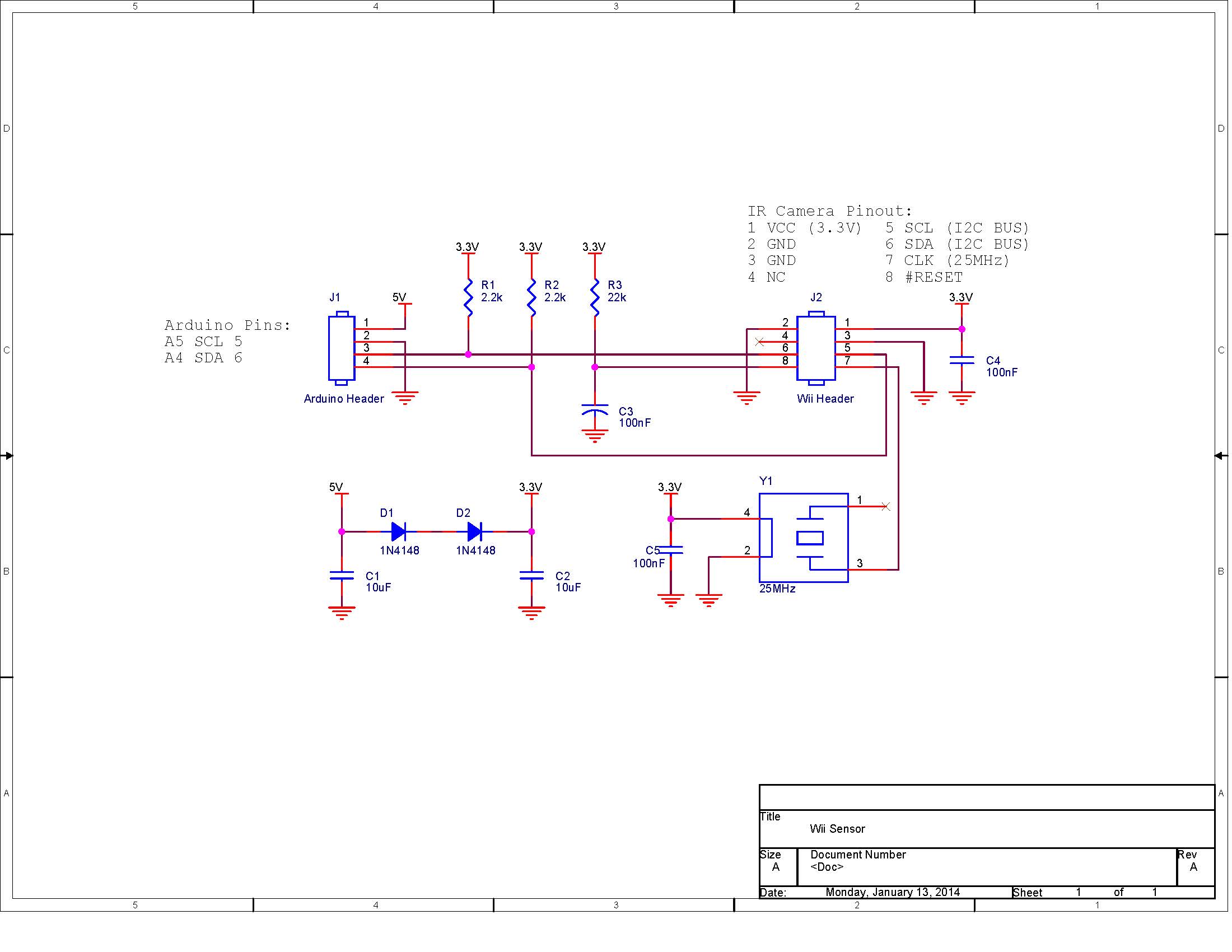
The system is designed to be manufactured as two separate components that when combined make up the complete MailBird delivery system. The first component is the landing sensor and algorithm which are simply manufactured on a printed circuit board to be attached to a number of autonomous delivery vehicles. A schematic of the landing module is included in Figure 1. The second component, the quadcopter requires human interaction to manufacture and test. Combining the two components using I/O ports on the Arduino based flight controller completes the MailBird delivery system. The final step for user compatibility requires developing delivery routes to fit each campus setting.

Figure 1 – Landing Module Schematic

Considerations when designing delivery routes on heavily populated campuses include avoiding densely foot traffic areas as well as developing delivery schedules around times with high pedestrian activity. The most dangerous aspect of the system to human safety is the four propellers approaching speeds of 1090rpm which provide lift and maneuverability for the MailBird. Propeller guards have been added into the quadcopter design in an attempt to prevent any incident.

Even though any unwarranted malfunction that would cause the quadcopter to fall out of flight could possibly be dangerous to the public the design chosen develops a simple, clean, safe image in the public eye. This image along with custom designed flight paths avoiding areas of high traffic and at the maximum altitude allowed by law will keep the MailBird out of a weary public eye.

Using the maximum allowed altitude not only eases the mind of the public, but it also cuts down on the noise pollution to the environment. Designed to fly at an altitude upwards of 400ft the MailBird delivery system is barely audible. The system is also 100% electric only requiring 33W per charge.

However, it is not only the minds of the public that must be convinced of its safety and ethics, the design is also largely shaped by politics and government regulations. Recently with allegations of the government’s invasion of privacy and the fear of drones being folded into this spying scheme there is a fine line to what robots can be designed to do. The MailBird design does not contain a video camera that could be used for unwanted data acquisition. The MailBird is designed to calculate accurate altitude information in order to abide by FFA airspace regulations and remain on a level legal for RC vehicles.

## Design Standards

GPS frequency/information protocol – GPS data transmitted over the L1 frequency band (1575.42 MHz), using NMEA 0183 data protocol.

Flight Height – *H.R. 658* *(FAA Air Transportation Modernization and Safety Improvement Act).* *SEC. 334 (b)* Standards for Operation and Certification – Not later than December 31, 2015, the Administrator shall develop and implement operational and certification requirements for the operation of public unmanned aircraft systems in the national airspace system. This will most likely be close to SEC. 334 (c) Agreements with Government Agencies (2) The agreements shall – (C) allow government public safety agency to operate unmanned aircraft weighing 4.4 pounds or less, if operated – (ii) less than 400 feet above the ground; (iii) during daylight conditions; (v) outside of 5 statute miles from any airport.

RC Frequency – Manual flight control transmitted over 2.4GHz spread spectrum.

# 3 Management Approach

Lab meeting times are Monday from 2-5 PM, Tuesday from 3-6 PM, and Wednesday from 3-5 PM.  The design process is divided into two, six-week long cycles.  Each cycle is divided into week-long management iterations.  Wednesday marks the beginning of each iteration, and the first part of Wednesday’s meeting will be to plan out what can be accomplished within the week.  At the end of the iteration, on Tuesday, we will generate a summary from the minutes and GitHub task tracking to create a weekly status report.  Decisions are made by majority consensus. At the beginning of each cycle, we will use the first meeting to plan out what needs to be done and then create associated tasks in our GitHub project. Each team member will be assigned tasks to complete by a given deadline.  During each meeting, we will take the first 10 minutes to describe what we’re trying to accomplish, and our intended solutions.  This will be a time for the team to collaborate and share ideas.  
  
GitHub will be used as both the primary method of communication and activity tracking. Tasks will be created and heavily organized so that there is a clear measure of progress and completion percentage.  Each task description will have criterion for task completion and the filepath for each associated file for the task. When work has been done on a task, a comment should be added stating progress and the description updated to show the status of the task.  Once all completion points are accomplished, the task should be closed.  
  
Minutes will be taken during the first 10 minutes of each meeting and then gleaned from the GitHub task comments.  This will keep our team updating the GitHub regularly and descriptively.

# 4 Budget

The most expensive component of the project is the quadcopter. As one of the team members already owns a custom-built quadcopter, he lent it to the team for the project, which significantly reduces costs. The costs of the project were therefore divided into a few general areas: the guidance module (detects the LEDs on the landing pad in order to precisely land the quadcopter), the landing pad (the base on which the quadcopter lands, which simulates a mailbox), the drop mechanism (used to carry mail), replacement parts for the quadcopter (to replace components that may be damaged while testing the senior design project), and a couple categories for other costs (such as shipping or extra parts). The estimated costs for the above categories came to approximately 72% of the available funds for the project (available funds calculated assuming a $50 contribution by each team member), which left 28% of the available funds for unforeseen expenses. A detailed view of the project budget can be seen in Appendix I.

# 5 Timeline

The proposal is to be completed by January 22. A working prototype of the guidance module camera shall be used during the proposal to demonstrate that the camera can detect infrared lights and position them relative to each other, an integral component of the precision landing portion of the project. A complete working prototype of the mail drone will be complete by the end of February. The rest of Cycle 1 (which ends March 6) consists of preparing the report and presentation for the cycle. Cycle 2 shall consist of finalizing the project, writing the user manual, and preparing the presentation and display for the Senior Design Fair. A detailed view of the project timeline can be seen in Appendix II.

# 6 Facilities Used

## Design Lab – Hardware construction

Development of MailBird’s hardware components will use lab facilities provided by Auburn University. Lab 368, a research lab provided by Dr. Roppel provides the necessary tools to manufacture and test the electrical circuits required to build the APELM. The lab provides design tools such as bread boards and soldering irons and expendables such as wire, resistors, capacitors etc. The lab will also serve as the primary location to store and assemble the quadcopter and APELM module. The MailBird team (Team 1) has recovered the first workbench on the left wall of 368.

## Labs 308 & 310

The computer labs provided by the Auburn University Electrical Engineering Department will be the primary location for software development, team meetings, document preparation, and presentation preparation. The labs provide fast computers with large monitors for group work and software development.

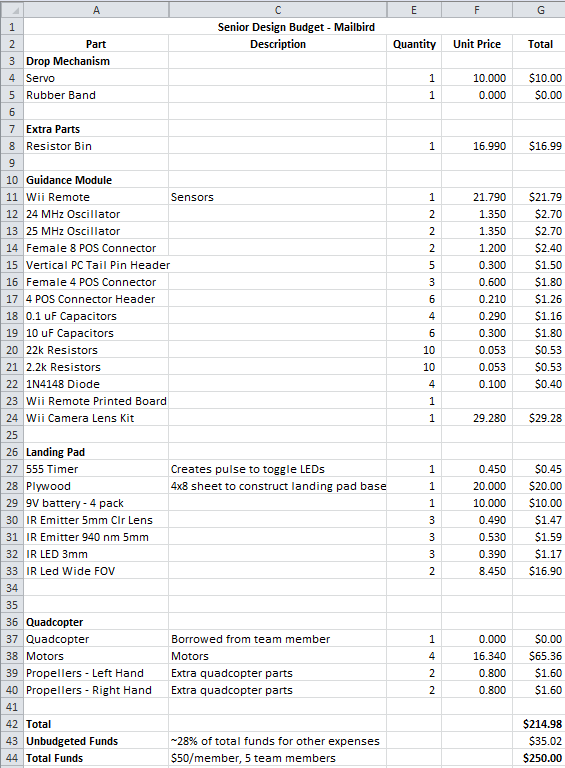
## Testing

During the initial design phases of MailBird, testing requiring flight by the quadcopter will take place outside. During development, the unpredictability of flight paths and the need for accurate GPS coordinates necessitate (for the safety of the craft and other people) that a wide open area be used. Later in the development process when the aircraft’s flight has become more stable and predictable the APELM system can be tested and refined inside Lab 368.

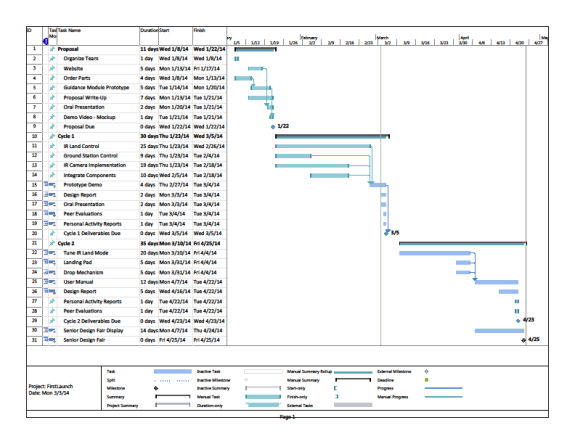
# 7 Disposition Agreement

A Disposition Agreement executed as of January 13, 2014 and effective as of the 1st day of May, 2014, by and between Auburn University and Hugh Dillon, Rick Holloway, Zac Hawkins, Ben Smith, Hunter Thorington was created to ensure the fair and proper transfer of all MailBird property at the end of the semester. This agreement can be found in Appendix III.

# Appendix I – Project Budget



# Appendix II – Project Gantt Chart



# Appendix III – Disposition Agreement

**DISPOSITION AGREEMENT**

This Disposition Agreement (this "Agreement"), executed as of January 13, 2014 and effective as of the 1st day of May, 2014, by and between Auburn University herein "Auburn” and Hugh Dillon, Rick Holloway, Zach Hawkins, Ben Smith, Hunter Thorington (herein "Team 1") In consideration of the mutual promises and covenants herein contained, the parties hereto agree as follows:

WHEREAS, Rick Holloway has agreed to lend full and unrestricted use and access to his personal custom model quad copter, herein “the Bird” to Team 1 for time period January 13, 2014 – May 1, 2014

WHEREAS, Auburn shall not retain any rights to the use of or access to Rick Holloway’s “the Bird”

WHEREAS, Team 1 shall accept as repayment for Rick Holloway’s generosity that all personal expenditures required for the development of “the Bird” shall be relinquished to Rick Holloway as personal property.

NOW THEREFORE, subject to the terms and conditions herein and acceptance by the Transferee Rick Holloway, and the Transferor Team 1 hereby agree to undertake the following actions as defined in Article I herein for the consideration stated herein.

ARTICLE I

TRANSFER OF THE BIRD

1.01

Immediately upon the execution of this Agreement the Transferor shall transfer the Bird to Rick Holloway.

1.02

The bird shall be free and clear of all liens and encumbrances and the Transferee shall have good title to the Bird immediately upon the execution of this Agreement.

By:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/s/ Thaddeus Roppel c/o Auburn /s/ Zac Hawkins

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/s/Hugh Dillon /s/ Ben Smith

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

/s/Rick Holloway /s/ Hunter Thorington

# Appendix IV – WiiCamera Library

\*/

WiiCamera.cpp - Arduino Library for the WiiCamera Sensor

This work was derived from;

1) http://www.kako.com/neta/2007-001/2007-001.html

2) www.stephenhobley.com

Sensor is conected to I2C port 0x58

Methods:

Init() : Initialization of I2C

Read() : Read sensor data and calculate Range

This function is optimized so the main host don't need to wait

You can call this function in your main loop

It returns a 1 if there is new data.

Internal functions:

Calculate() : Calculate Range in real units

\*/

// Includes //////////////////////////////////////////////////////////////

#include "APM\_WiiCamera.h"

#include <Wire.h>

// Private Methods //////////////////////////////////////////////////////////////

void WiiCamera**::**Write2bytes**(**byte d1**,** byte d2**)**

**{**

Wire**.**beginTransmission**(**IRslaveAddress**);**

Wire**.**write**(**d1**);**

Wire**.**write**(**d2**);**

Wire**.**endTransmission**();**

delay**(**10**);**

**}**

// Constructors ////////////////////////////////////////////////////////////////

WiiCamera**::**WiiCamera**()**

**{**

WiiCamera\_State **=** 0**;** // Initial state

idx **=** 0**;**

blobcount **=** 0**;**

Blob**[**0**].**number **=** 1**;**

Blob**[**1**].**number **=** 2**;**

Blob**[**2**].**number **=** 3**;**

Blob**[**3**].**number **=** 4**;**

**}**

// Public Methods //////////////////////////////////////////////////////////////

// init the Wii camera

void WiiCamera**::**init**()**

**{**

IRsensorAddress **=** 0xB0**;** // PixArt wii camera i2c address

IRslaveAddress **=** IRsensorAddress **>>** 1**;** // This results in 0x58 as the address to pass to TWI

Wire**.**begin**();**

// Wii IR sensor initialize

// http://wiki.wiimoteproject.com/IR\_Sensor

// Level 1: p0 = 0x72, p1 = 0x20, p2 = 0x1F, p3 = 0x03

// Level 2: p0 = 0xC8, p1 = 0x36, p2 = 0x35, p3 = 0x03

// Level 3: p0 = 0xAA, p1 = 0x64, p2 = 0x63, p3 = 0x03

// Level 4: p0 = 0x96, p1 = 0xB4, p2 = 0xB3, p3 = 0x04

// Level 5: p0 = 0x96, p1 = 0xFE, p2 = 0xFE, p3 = 0x05

// p0: MAXSIZE: Maximum blob size. Wii uses values from 0x62 to 0xc8

// p1: GAIN: Sensor Gain. Smaller values = higher gain

// p2: GAINLIMIT: Sensor Gain Limit. Must be less than GAIN for camera to function. No other effect?

// p3: MINSIZE: Minimum blob size. Wii uses values from 3 to 5

Write2bytes**(**0x30**,**0x01**);** //Control byte, allows modification of settings

Write2bytes**(**0x06**,**0x90**);** // MAXSIZE - Maximum blob size. Wii uses values from 0x62 to 0xc8.

Write2bytes**(**0x08**,**0xC0**);** // GAIN - Sensor Gain. Smaller values = higher gain. Numerical gain is proportional to 1/2^(n/16) for n<0x40

Write2bytes**(**0x1A**,**0x40**);** // GAINLIMIT - Sensor Gain Limit. Must be less than GAIN for camera to function. No other effect?

Write2bytes**(**0x33**,**0x33**);** // MODE - Camera mode

Write2bytes**(**0x30**,**0x08**);** // was 2nd - suspect it really needs to be here

delay**(**100**);**

WiiCamera\_State **=** 1**;** // Ready to Read state

**}**

// Read the sensor.

int WiiCamera**::**read**()**

**{**

int data\_buf**[**16**];**

//int idx=0;

int s**;**

**if(**WiiCamera\_State**==**1**)** // New Read request

**{**

// Set Wii Camera to correct data mode

Wire**.**beginTransmission**(**IRslaveAddress**);**

Wire**.**write**(**0x36**);**

Wire**.**endTransmission**();**

// request data from sensor

Wire**.**requestFrom**(**IRslaveAddress**,** 16**);**

// clear space for new data

**for** **(**idx**=**0**;**idx**<**16**;**idx**++)** data\_buf**[**idx**]=**0**;**

// index fo data pointer

idx **=** 0**;**

// Reset the blob counter

blobcount **=** 0**;**

WiiCamera\_State**++;** // new state expects data

**}**

**if** **(**WiiCamera\_State**>=**2**)** // Waiting for first Data byte from a previous read request

**{**

WiiCamera\_State**++;** // new state, now reading data

**while(**Wire**.**available**()** **&&** idx **<** 16**)**

**{**

data\_buf**[**idx**]** **=** Wire**.**read**();**

idx**++;**

**}**

// if we have 16 bytes of data

**if** **(**idx **>=** 16**)**

**{**

Blob**[**0**].**X**=** data\_buf**[**1**];**

Blob**[**0**].**Y **=** data\_buf**[**2**];**

s **=** data\_buf**[**3**];**

Blob**[**0**].**X**+=** **(**s **&** 0x30**)** **<<**4**;**

Blob**[**0**].**Y **+=** **(**s **&** 0xC0**)** **<<**2**;**

Blob**[**0**].**Size **=** **(**s **&** 0x0F**);**

**if** **(**Blob**[**0**].**Size**<**15**)** blobcount**++;**

Blob**[**1**].**X **=** data\_buf**[**4**];**

Blob**[**1**].**Y **=** data\_buf**[**5**];**

s **=** data\_buf**[**6**];**

Blob**[**1**].**X **+=** **(**s **&** 0x30**)** **<<**4**;**

Blob**[**1**].**Y **+=** **(**s **&** 0xC0**)** **<<**2**;**

Blob**[**1**].**Size **=** **(**s **&** 0x0F**);**

**if** **(**Blob**[**1**].**Size**<**15**)** blobcount**++;**

Blob**[**2**].**X **=** data\_buf**[**7**];**

Blob**[**2**].**Y **=** data\_buf**[**8**];**

s **=** data\_buf**[**9**];**

Blob**[**2**].**X **+=** **(**s **&** 0x30**)** **<<**4**;**

Blob**[**2**].**Y **+=** **(**s **&** 0xC0**)** **<<**2**;**

Blob**[**2**].**Size **=** **(**s **&** 0x0F**);**

**if** **(**Blob**[**2**].**Size**<**15**)** blobcount**++;**

Blob**[**3**].**X **=** data\_buf**[**10**];**

Blob**[**3**].**Y **=** data\_buf**[**11**];**

s **=** data\_buf**[**12**];**

Blob**[**3**].**X **+=** **(**s **&** 0x30**)** **<<**4**;**

Blob**[**3**].**Y **+=** **(**s **&** 0xC0**)** **<<**2**;**

Blob**[**3**].**Size **=** **(**s **&** 0x0F**);**

**if** **(**Blob**[**3**].**Size**<**15**)** blobcount**++;**

WiiCamera\_State **=** 1**;** // back to ready to Read state

**}**

**if** **(**WiiCamera\_State**>=**10**)** WiiCamera\_State **=** 1**;** // Waited too long for data - back to ready to Read state

**}**

**return(**blobcount**);**

**}**