Concordia University

**Department of Computer Science**

**and Software Engineering**

**Embedded Systems**

**SOEN 422/2 --- December 2012--- Section MM**

**Project Final Report**

***This report should cover all phases of the project, including a detailed description of the project plan, the state of the implementation of the project at delivery time (which is unlikely to be complete, detailed report of the hardware and the software of the unit produced, and a detailed report of the development support system. The report may include any or all of the first proposal report, and any reference material copied and quoted which ill illustrate the components or development systems used.***

***The document layout and physical presentation shall count for 15% of the value of marks of the proposal.***

|  |  |
| --- | --- |
| **Team information** | |
| **Team Number** | |
| **Member Name** | **Member SID** |
| Jeremy Pallats | 9619550 |
| Jaspreet Singh Lidder | 9080449 |
| Ralph d'Almeida | 9464131 |

Table of Contents

[1. Introduction 4](#_Toc343541297)

[2. Project Description 4](#_Toc343541298)

[2.1 Project Purpose 4](#_Toc343541299)

[2.2 Unit function and performance 4](#_Toc343541300)

[2.3 Project Hardware 4](#_Toc343541301)

[2.4 Project Software 5](#_Toc343541302)

[2.5 Scope of Completion 5](#_Toc343541303)

[3. Hardware Design 5](#_Toc343541304)

[3.1 System Design 5](#_Toc343541305)

[3.1.1 High Level System Overview 6](#_Toc343541306)

[3.1.2 Low Level System Overview 6](#_Toc343541307)

[3.2 Subsystem Design 7](#_Toc343541308)

[3.3 System Intercommunication 8](#_Toc343541309)

[4. Software Design 8](#_Toc343541310)

[4.1 System Software design 8](#_Toc343541311)

[4.2 Software Subsystem Designs 9](#_Toc343541312)

[4.3 Arduino functions 9](#_Toc343541313)

[4.4 System Software Communication 10](#_Toc343541314)

[4.4.1 Patriot Wireless 10](#_Toc343541315)

[4.4.2 XBee Bluetooth 10](#_Toc343541316)

[5. Operating System and Development Software 10](#_Toc343541317)

[5.1 Initial OS Installation 10](#_Toc343541318)

[5.2 Initial Internet Sharing with Windows 10](#_Toc343541319)

[5.3 Partition Resizing 11](#_Toc343541320)

[5.4 SSH Login 11](#_Toc343541321)

[5.5 Development Packages 12](#_Toc343541322)

[5.6 Wireless USB Patriot Stick 12](#_Toc343541323)

[5.6 Python Bluetooth Setup 13](#_Toc343541324)

[5.7 MJPG\_Streamer 13](#_Toc343541325)

[5.8 nGear G-H508 Mini 4 USB Hub 13](#_Toc343541326)

[6. System Performance 14](#_Toc343541327)

[6.1 Component testing 14](#_Toc343541328)

[6.2 System test 14](#_Toc343541329)

[7. System Delivery 14](#_Toc343541330)

[8. Project Process discussion 15](#_Toc343541331)

[8.1 First Phase – Teensy++ and C Code 16](#_Toc343541332)

[8.1.1 Description 16](#_Toc343541333)

[8.1.2 Difficulties 16](#_Toc343541334)

[8.1.3 Successes 16](#_Toc343541335)

[8.2 Second Phase – Teensy++, Rover Motors and C Code 16](#_Toc343541336)

[8.2.1 Description 16](#_Toc343541337)

[8.2.2 Difficulties 16](#_Toc343541338)

[8.2.3 Successes 16](#_Toc343541339)

[8.3 Third Phase – Beagle Board, Rover Motors and C Code 16](#_Toc343541340)

[8.3.1 Description 16](#_Toc343541341)

[8.3.2 Difficulties 16](#_Toc343541342)

[8.3.3 Successes 17](#_Toc343541343)

[8.4 Fourth Phase – ArduCopter, Beagle Board, C Code and Wireless Communication 17](#_Toc343541344)

[8.4.1 Description 17](#_Toc343541345)

[8.4.2 Difficulties 17](#_Toc343541346)

[8.4.3 Successes 17](#_Toc343541347)

[8.5 Project Overview 17](#_Toc343541348)

[9. Appendix 18](#_Toc343541349)

[9.1 Software Source Code 18](#_Toc343541350)

[9.1.1 ChangePWM.py 18](#_Toc343541351)

[9.1.2 Start\_stream.sh 19](#_Toc343541352)

[9.1.3 In Progress Bluetooth Code 19](#_Toc343541353)

[10. References 21](#_Toc343541354)

Table of Figures

[Figure 1: High Level System Overview 6](#_Toc343534571)

[Figure 2: Low Level System Overview [1] 6](#_Toc343534572)

[Figure 3: Subsystem Design [2] 7](#_Toc343534573)

[Figure 4: ArduPilot XBee Adapter [2] 7](#_Toc343534574)

[Figure 5: System Intercommunication 8](#_Toc343534575)

[Figure 6: Resized Partitions on Beagle 11](#_Toc343534576)

[Figure 7: Direct SSH Login to Beagle 12](#_Toc343534577)

[Figure 8: Configured /etc/network/interfaces File 13](#_Toc343534578)

[Figure 9: Project Process 15](#_Toc343534579)

# Introduction

This document provices an overview of the ArduCopter project for SOEN 422. This document presents the interesting challenge of building an embedded system to control and operate the ArduCopter. Our system was to involve two embedded computers, a Teensy to provide interfacing and low level drive functions, and a Beagle to provide any intelligence the system may need. In the later stages, it was discovered that we would be able to completely operate the ArduCopter via the Beagle by directly plugging the control wires into PWM expansion slots. Therefore, the Teensy has been completely subsumed by the Beagle. There must also be the ability to send commands over a Bluetooth interface. We will first describe the system under investigation; provide hardware and software overviews and development process information and close by discussion of the implementation issues.

# Project Description

This shall consist of a description of a complete version of the project as it would exist if you had the time and resources to complete it. The description will describe what the unit produced is, what it does, and what its expected performance is in the following sections. (This is not the design section).

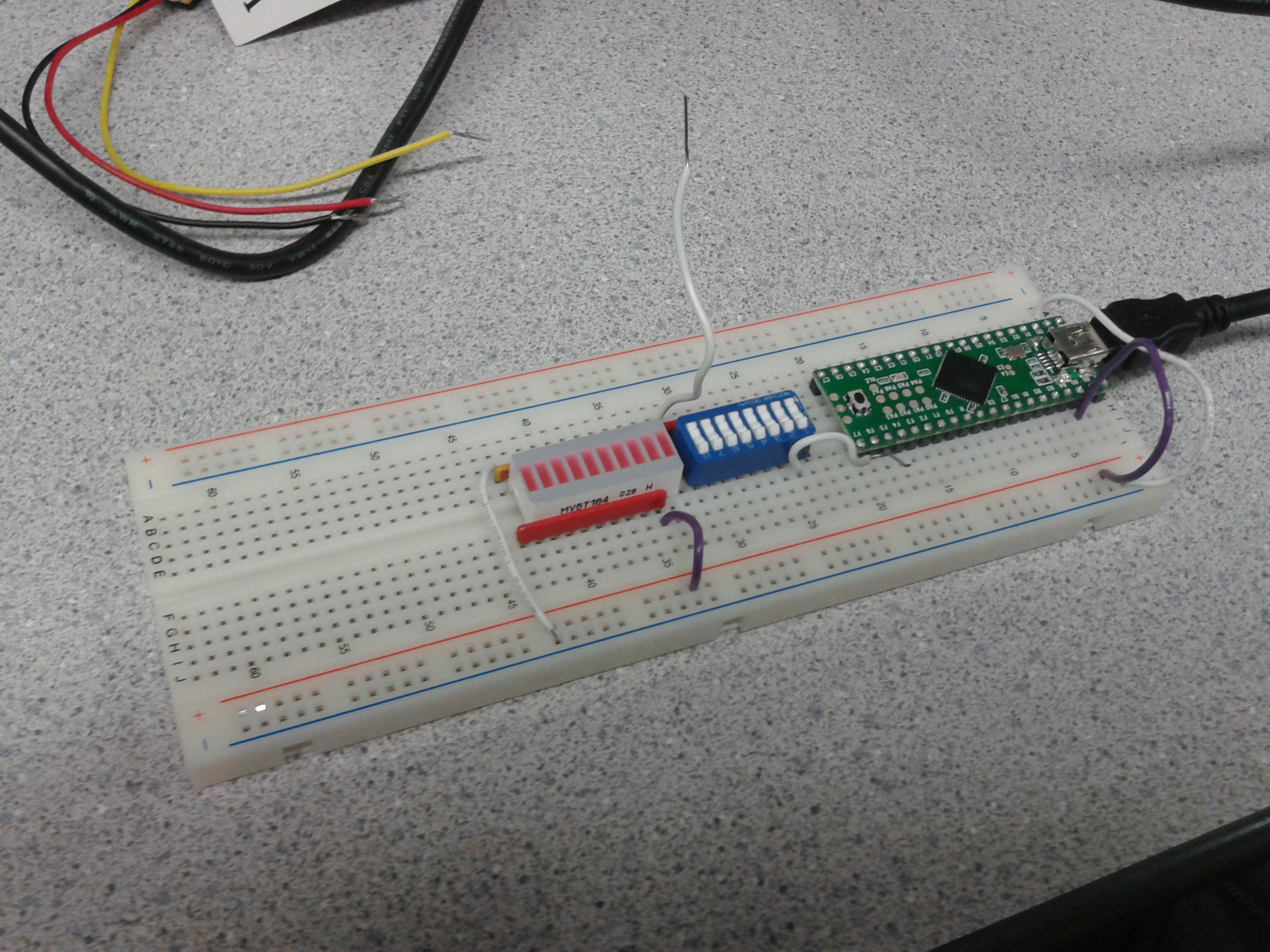
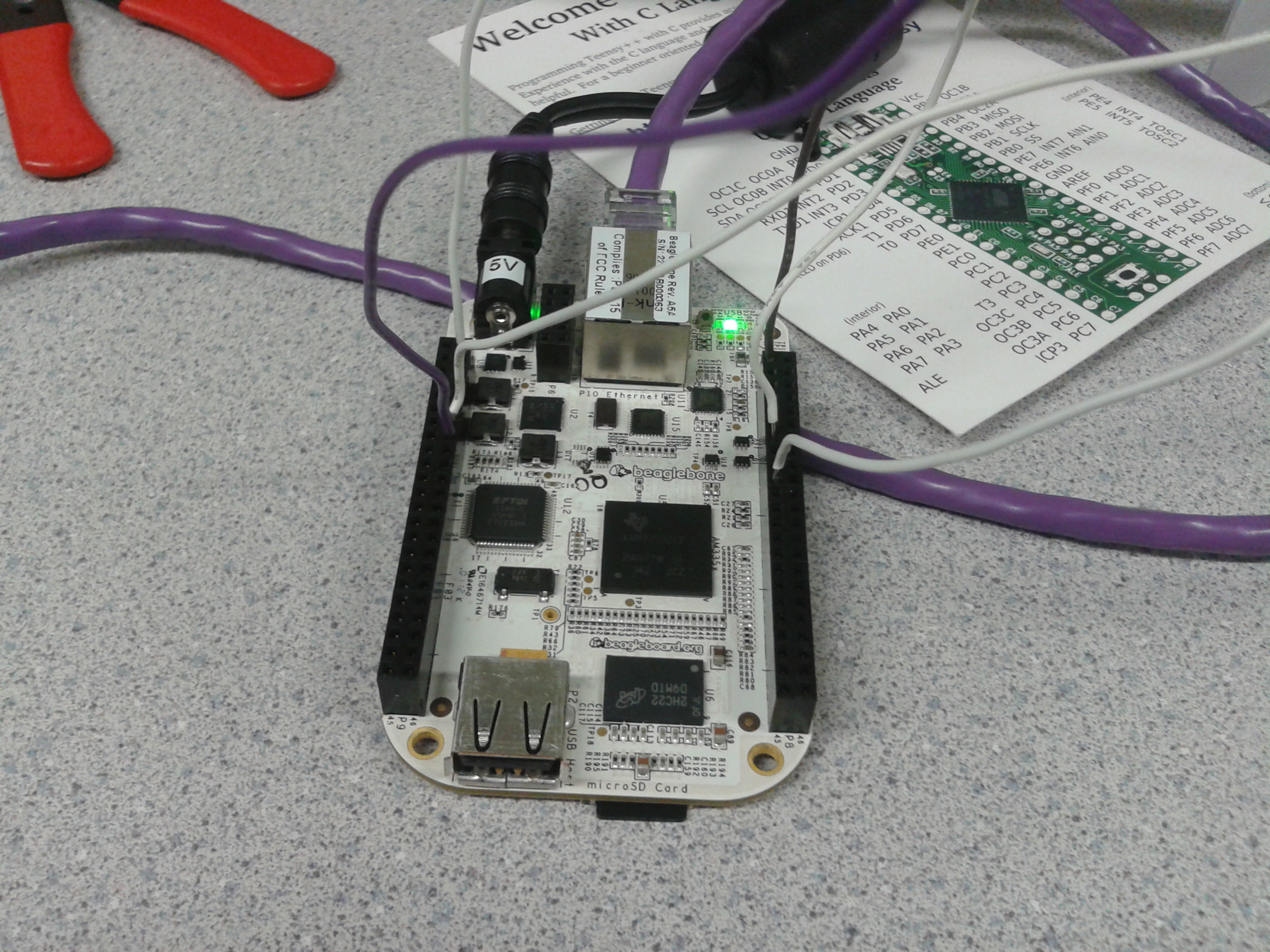
## 2.1 Project Purpose

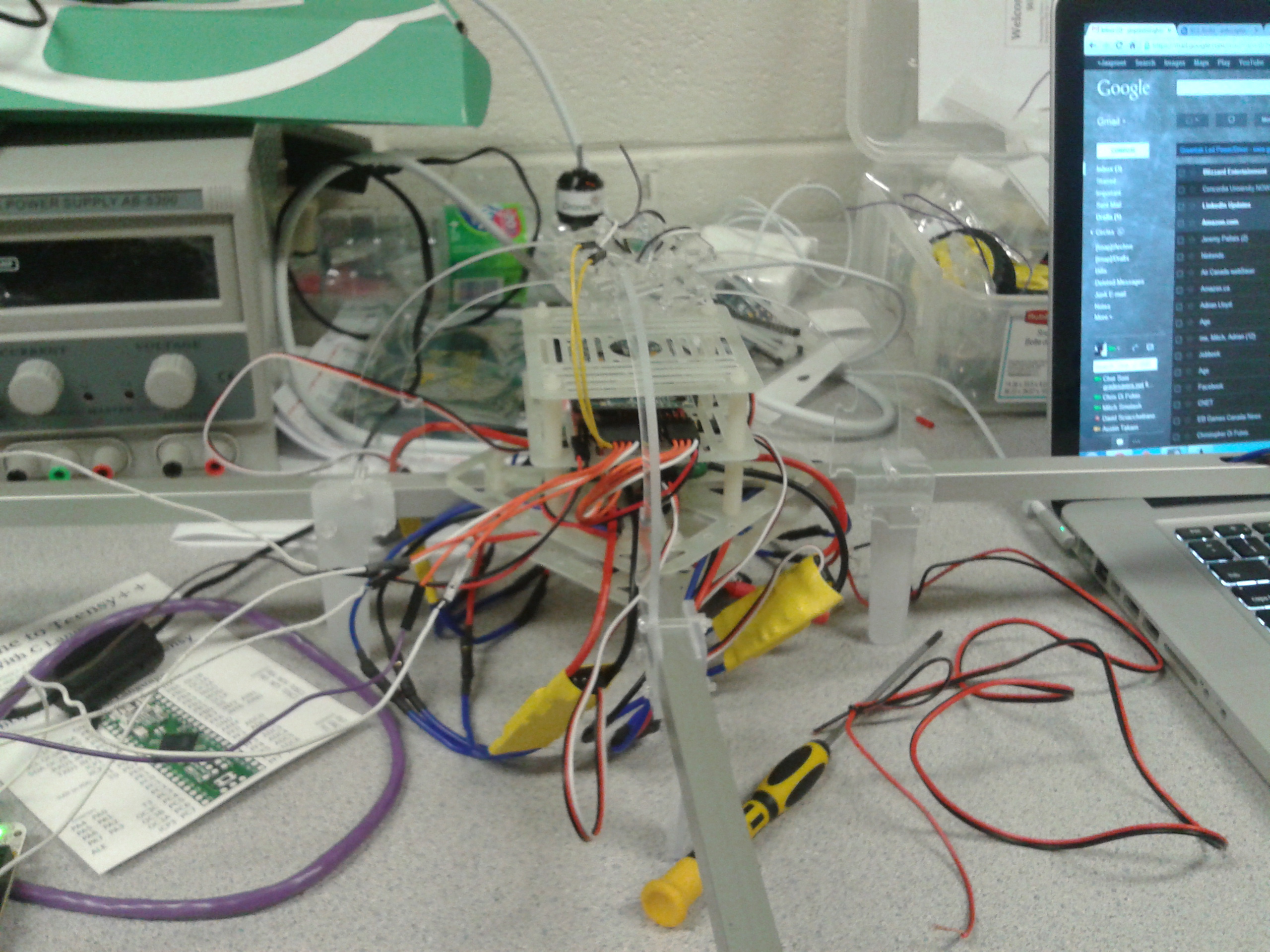
The unit is the ArduCopter kit that the team assembled and set up for a test flight demonstration with the Beagle controlling the operation of the machine. Commands were to be sent over Bluetooth. The user was to be able to get video information of the current whereabouts via a streamed webcamera.

## 2.2 Unit function and performance

The units altogether will be the flight control system of the ArduCopter; that is, they will control the directions and the speed the motors move as well as the lift up and landing. Hence, the units are set for autolevel and auto altitude control as automatic takeoff and landing. With a 2200 battery pack, we expected 10 minutes autonomy fly at 50km/h speed with 100% throttle.

## 2.3 Project Hardware

* Teensy++ 2.0 board, model AT90USB1286 (and all related components/wiring)
* Beagle Board (serve as controller via PWM signals)
* ArduCopter
  + ArduPilot Mega
  + ArduCopter Frame (or multi-rotor frame)
  + Motors/ESC/Propellers
  + Radio Transmitter/Reciever
  + Battery and Charger



* Bluetooth components (XBee Chips)
* Two USBA to USB mini connectors (USB Serial connection with laptops)
* Wireless Patriot USB stick (provide Wi-Fi connection to beagle board)
* 5V DC power for the board. (power)
* Ethernet cable for internet sharing
* Notebook with Linux/Windows 7 installed
* Logitech Webcamera
* nGear G-H508 Mini 4 USB Hub

## 2.4 Project Software

The project runs on the Ubuntu platform running on the Beagle. This is the main controller, several libraries are added to facilitate communication between the user and the device. These are explained the section 5 where the overall platform is documented as well as the steps required to set it up. Additionally, the ArduPilot Mega software was used for initial setup and some testing of the ArduCopter. It would have been nice had we the optional GPS unit to test with, that may prove to be a good addition for subsequent semesters.

## 2.5 Scope of Completion

* The ArduCopter was completely assembled without many difficulties.
* Basic operation of all the motors via a python script on the Beagle was completed.
  + A small amount of additional fine tuning to the values used in the PWM to control the motors would have been a welcome addition.
* The Wireless communication was completed satisfactorily though the Patriot stick proved to be somewhat unreliable.
* The Webcamera was tested and verified to work. Streaming software was loaded onto the machine working well over an established wireless connection.
* Work was commencing on the Bluetooth controller but did not proceed far enough due to lack of time. We have included all that was completed.

# Hardware Design

This section gives a description of the detailed hardware design of the unit in question. It provides an overall view of what would be required to make the ArduCopter fly.

## 3.1 System Design

The hardware system shall be presented using a block diagram and be given in two perspectives (high and low). The Overview of the system can be presented in a high and low level view.

### High Level System Overview

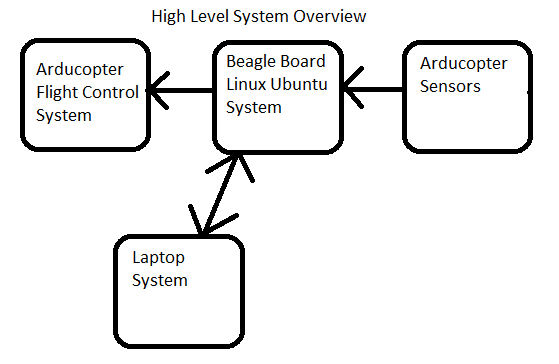


Figure 1: High Level System Overview

### Low Level System Overview

The following low level diagram was found on the internet, which provides a very similar low level system overview of our ArduCopter system

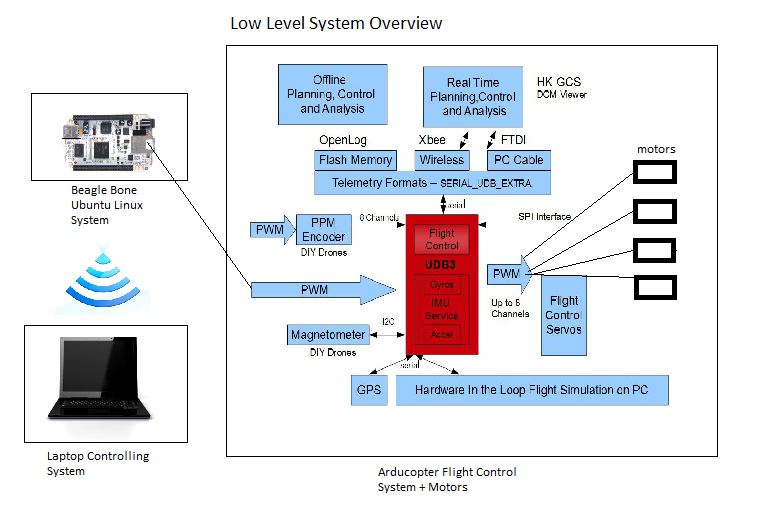


Figure 2: Low Level System Overview[1]

## Subsystem Design

The subsystem view provides detailed design of the internals of the systems, such as circuitry layout.

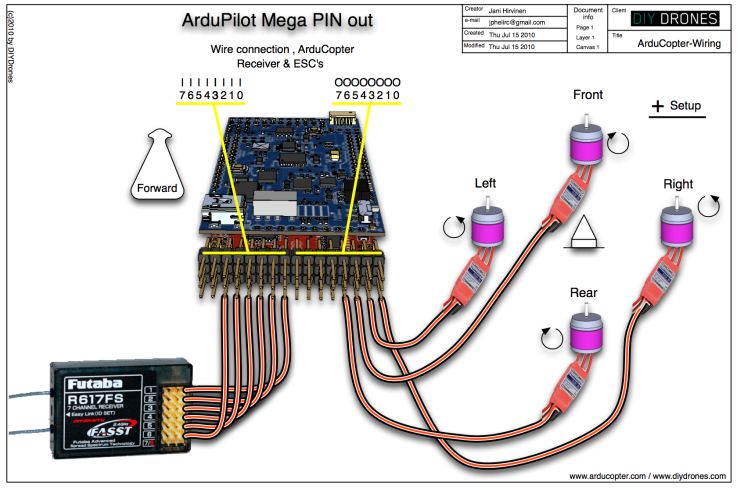


Figure 3: Subsystem Design[2]

Illustrated cable connections between APM + Receiver, APM + ESC's and motor locations for ArduCopter

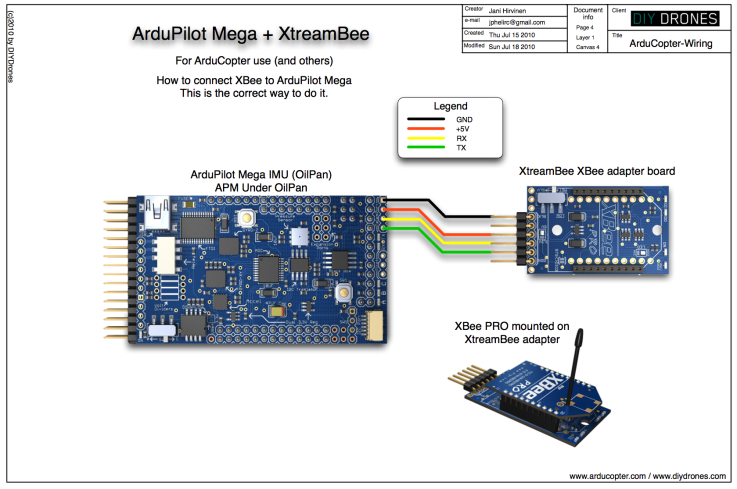


Figure 4: ArduPilot XBee Adapter[2]

Illustration of how to connect XtreamBee XBee adapter to OilPan IMU board

## 3.3 System Intercommunication

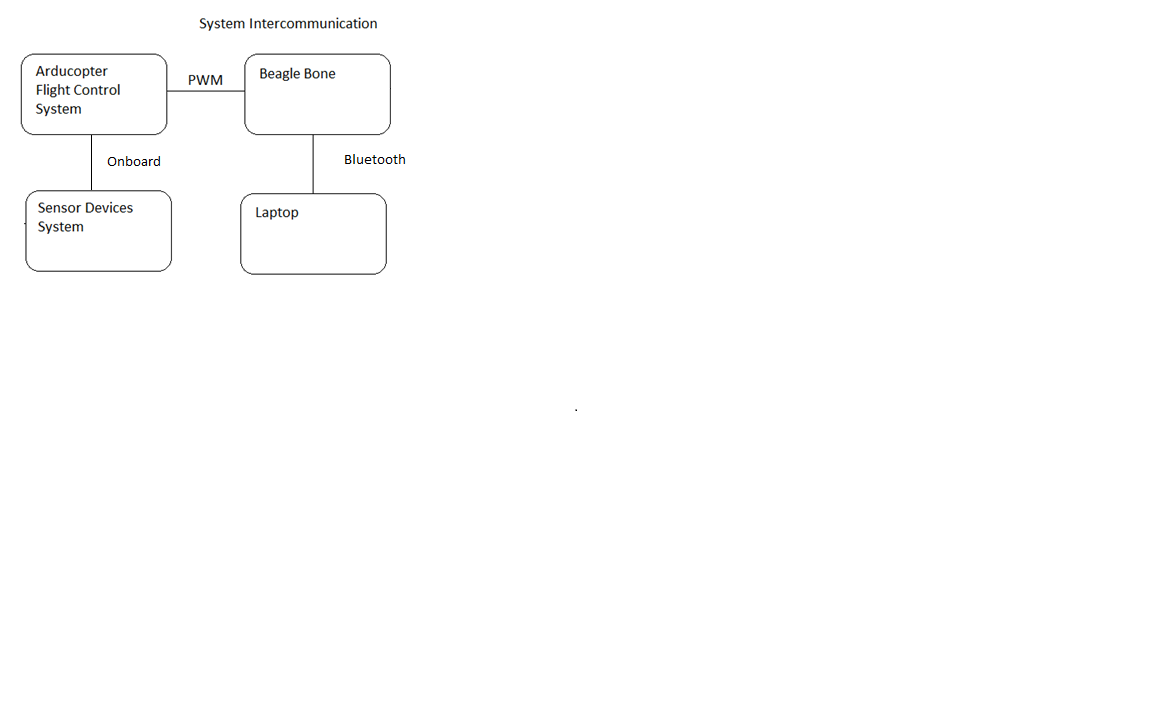


Figure 5: System Intercommunication

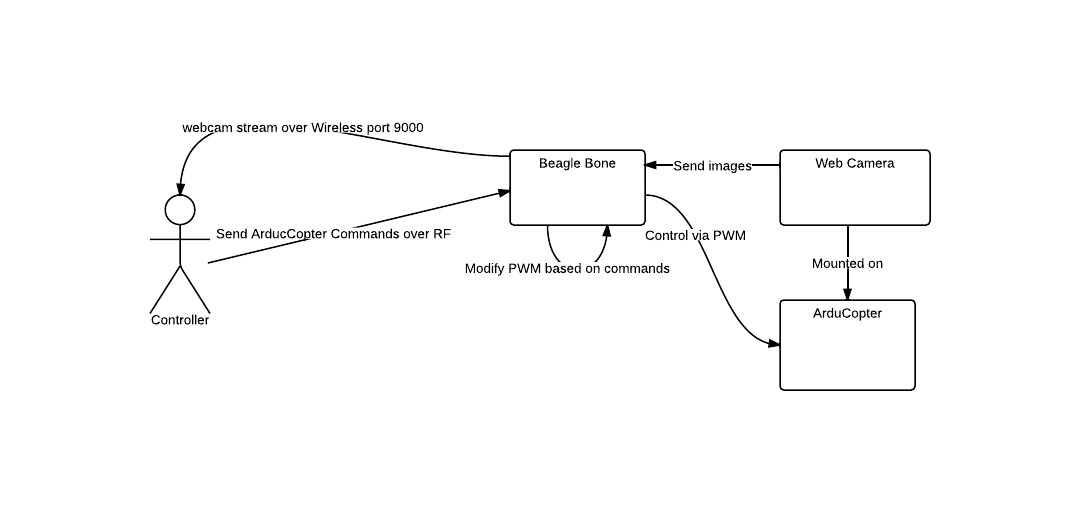
# Software Design

In the following sections, the overall software design will be elaborated upon from a high enough level as to specify the software platform and the major components running on them to drive the copter.

## System Software design

We did not employ any low level software in order to make the ArduCopter work. The operating system installed was the stock Ubuntu arm image provided by the department. In order to make several components work a number of packages were additionally installed; these are detailed in section 5 of this report.

## Software Subsystem Designs



There are two major software modules in the design listed above, we have in the system: the web camera streaming software mostly handled by mjpg\_streamer project and the python control code manipulating the Beagle. I’ll describe these in turn.

The MJPG\_Streamer project was a very interesting C capture program discovered while searching for a simple way to stream. Initially the team attempted to use VLC, this proved far too taxing though as VLC is mainly designed for streaming encoded data. Encoding the RAW stream from the webcamera was far too taxing for the device and would never work to stream images live. It was discovered during this investigation that Linux V4L drivers allow direct access to capturing JPEG images from the webcamera. The MJPG\_Streamer project is a simple C program that capturs frames through the V4L driver and transmitts them over a web server to be streamed by any user that can connect by browser.

We tested the software and found it to be suitable. We wrote a small start\_stream.sh script that starts the streaming software on port 9000. The data is transmitted over the Wireless connection since it is too bandwidth intensive to use the Bluetooth interface. This server persists even if the wireless connection drops and is restored, so that we can reconnect the stream even if it drops while continuing to be in control of the helicopter via the XBee.

The PWM control for the device is designed to come from a single file called ChangePWM.py, it would finally be integrated with a Bluetooth server in order to receive commands encoded in a very simple format so that a single simple command could modify the entire attitude of the helicopter. We did not complete the final integration with Bluetooth code so at the moment the ChangePWM.py script merely accepts commands via arguments at invocation.

Both the Bluetooth communication system and the MJPG\_Streamer software rely on a client-server architecture to receive commands and transmit information based on requests. In the final setup, the webcamera would be available for access via port 9000 on the Beagle (accessible directly via a fixed IP over wireless). The Bluetooth server would sit on the Beagle awaiting connections from clients to transmit short strings of characters carrying a simple format agreed upon to convey the 4 different PWM values to manipulate the different flight characteristics of the ArduCopter.

## Arduino functions

None were used in creating the software to drive the ArduCopter.

## System Software Communication

Communication with the Beagle Bone occurs via two different networking chips. The Bluetooth XBee chips and the Wireless Patriot USB Stick.

### Patriot Wireless

Patriot Wireless was used to transmit the JPEG images. This was required since Bluetooth is not designed for large bandwidth intensive transmissions. The MJPG\_Stream software creates a webserver on a designated port that can be directly accessed in a browser. This then allows a user to watch a stream that is composed purely of JPEGs sent at a given framerate in the browser. No sound is transmitted in the current implementation.

### XBee Bluetooth

Bluetooth is used to communicate with the Beagle from a given computer. The client starts a sending program that connects to the XBee connected to the Beagle running a Bluetooth server waiting for connections. It then transmits a single line at a time that contains as many commands as a user wishes to modify. The format decided on was simple each of the four PWM values for roll, throttle, yaw and pitch could be provided in a single line by packing them in the following format: r-190,t-190,y-190,p-190. Note that a user could pass any arbitrary value at this point. A possible augmentation would be to send percentages and then convert those server side into the correct PWM based on the final ArduCopter tuned PWM values.

# Operating System and Development Software

In this section, all of the steps required in order to make the operating system ready for development and supporting the above modules has been listed in their own sections. These configurations and packages were essential to getting wireless connectivity, getting Bluetooth setup and then developing software directly on the Beagle itself. This section is broken into sections that describe all steps taken in the sequence they were done.

## Initial OS Installation

This phase was relatively straightforward for our team. One of our team members had a fully functional Ubuntu install on his notebook with an SD card reader. In order to write the image, we simply took the micro SD card and put it into the SD Adapter. Then we were successfully able to write to the device with image prepared earlier. Once this task was completed, it was simply a matter of inserting the SD card into the Beagle and connecting the power and USB cable to be able to verify.

In order to verify, some steps were required on our Windows PC. First, it was necessary to install the Beagle Board USB driver on Windows 7. This was handled automatically by the driver manager. In order to be able to use putty over a COM device, a further driver was necessary in order to translate the Beagle USB to COM. In searching the internet, we came across the Prolific USB-to-Serial[3]. This driver mapped the Beagle to a COM virtual port. We were then able to configure putty with the appropriate COM settings as pictured below to connect as seen in the second picture.

## Initial Internet Sharing with Windows

With a working image of Ubuntu and the able to connect over the COM it now became necessary to ensure the device had appropriate internet connectivity in order to install needed packages.

In order to setup the initial internet connectivity, we followed a tutorial available from Microsoft's support website that detailed how [4] to create a shared internet connection. In the end, the LAN port of our notebook shared the wireless internet from our existing connection with the the Beagle connected via LAN. This was successfully tested by performing a ping to one of Google’s public DNS servers.

With this done, we proceeded to install a series of packages and configure the device to be setup for development.

## Partition Resizing

It was noted that the root partition wasn’t very large on the SD card. It was using only 1.7 GB out of 3.6. By rebooting to our notebook with Ubuntu and inserting the SD card with adapter in the memory card reader we were able to resize the partition to take up all unused space. This provides us with much more space for installing extra packages under root. The new resized partition can be seen with the following image:

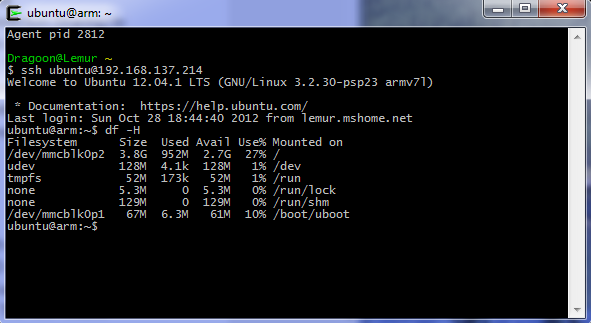


Figure 6: Resized Partitions on Beagle

## SSH Login

Since we do not always want to rely on a COM driver on windows, we chose to install the openssh-server package and all other necessary SSH components on the Beagle. This allows us to connect to the Beagle over port 22 by logging in with the user/password combination of the default user. In order to further streamline this, we generated a RSA key using ssh-keygen and the default parameters. This public key was then added to the list of authorized keys for the system. This now allows for easy authentication when logging in by simply checking the key.



Figure 7: Direct SSH Login to Beagle

## Development Packages

In order to have the complete development setup on this machine, we installed the following development packages:

* **build-essential:** A meta-package that triggers installation of the main build tools such as gcc, make and libc.
* **avr-libc:** The avr specific c library files necessary for compilation.
* **gcc-avr:** The avr specific gcc variant.
* **gdb-avr:** Avr variant of the CLI gnome debugger.
* **avrdude:** Command line flash program to remotely program the teensy later.
* **binutils-avr:** The standard binary utilities but avr specific.
* **python:** The standard python library (version 2.7) that comes installed was used later.
* **bluez:** The Bluetooth library and python bindings were installed on this machine and the controller.

With all of these packages downloaded and installed, we cloned our repository from GITHub to a folder under our /home/ubuntu user. We verified that these tools were working correctly by compiling with our Makefile a part of lab2 code. We also verified ubuntu by starting the interactive python prompt.

## Wireless USB Patriot Stick

The stick did not work when first connected and after some quick searching, it was discovered that a simple change was required. According to this site [5] on Ubuntu one simply needs to make a complete copy of the existing realtek RTL8192SE driver under /lib/firware. We used the stock command provided in the tutorial as below:

**sudo cp -R /lib/firmware/RTL8192SE /lib/firmware/RTL8192S**

With this step done, we were able to test the USB wireless stick by following the steps outlined in the elinux BeagleBoard wiki[6]. We edited the /etc/network/interfaces file and simply added the ssid and psk for a known network. This device then connected and without the LAN connection we were able to ping Google’s public DNS servers. Note that we tested a WPA-PSK network, additional testing will be needed to connect via the tunnelled authentication method at Concordia. It might alternatively be good to simply bring in a router and use that as a repeater for PSK from an ethernet line in the lab.

With both the wireless stick and the SSH server running it is now easy to remotely login to the Beagle without a physical USB connection or Ethernet cable so long as the IP assigned to the Beagle is known or fixed in some fashion. This wireless connection will be used for streaming the web camera. The more reliable short range Bluetooth connection will be used for sending commands to the helicopter.

auto lo

iface lo inet loopback

# The primary network interface

auto eth1

iface eth1 inet dhcp

# Example to keep MAC address between reboots

#hwaddress ether DE:AD:BE:EF:CA:FE

# WiFi Example

auto wlan0

iface wlan0 inet dhcp

wpa-ssid "SSID"

wpa-psk "Passphrase"

Figure 8: Configured /etc/network/interfaces File

## 5.6 Python Bluetooth Setup

In order to setup the Bluetooth libraries to be able to easily communicate with the XBee device from the Beagle Bone, we setup a free open source library called PyBBIO that is a library for the development of Bluetooth communication software via devices like the XBee. The software was downloaded from GITHub [7] onto the Beagle Bone and then installed by simply running the setup.py script with elevated administrative privileges. We tested that this was properly installed by executing a simple blinking program that made one of the board’s LEDs blink.

## MJPG\_Streamer

The streaming software we relied on to send webcam video to the controller came from an existing SourceForge project [8] that could be customized to our needs. No pre-existing binaries were found that were compatible with the ARM processor, so the source code had to be built on the device and installed into a local folder that could be accessed as needed. The start\_stream.sh described in the source code appendix is all that is needed to start streaming.

## 5.8 nGear G-H508 Mini 4 USB Hub

During the project, we realized we needed at least two USB ports, one for the Wireless Stick and another for the camera. While soldering on another USB port as had been seen on the internet and connecting it via the expansion wires would have been more educational, due to time constraints it was decided to be pragmatic and find an existing solution. Most USB hubs require power but this small mini 4 port hub is designed for low drain devices and it was ordered. It was tested and does support the Webcamer and Wireless in simultaneous operation.

# System Performance

## 6.1 Component testing

Each individual component had a form of manual testing performed upon it with input and output validations. For example, when we tested our C serial interface commands, we tried using complex text queries or false values in order to check the response. In terms of components such as the Beagle Board, certain tests such as bootup, communication with external devices and network activeness were tested. Software components such as the web server running on the beagle to broadcast the web cam was tested in terms of network access and server status.

## 6.2 System test

The different components of the system formed the entirety of the project. A form of integration testing was performed here, where we would test specific component behaviours when it worked in conjunction with other parts.

# System Delivery

This section provides instructions as to how to set up and use the unit, assuming that all of the code and development of features that were in progress was completed.

User Manual

7.1 System initialisation

In order to use the system, the Beagle must be connected to both the Webcamera and the Patriot Wireless stick. Ensure that the proper information is in the interfaces file to get a wireless connection. Once this is done, the webcamera stream can be activated by using the start\_stream.sh script available on the path of the device. Next simply start the XBee and startup the then completed ChangePWM.py file in order to ready the device to receive commands.

7.2 System operation

A client then uses a simple command format as described to arm and start the helicopter. Once started, he could open a browser and get a stream of video that would tell him what he was encountering while sending commands over the Bluetooth connection from a simple command interface to the device.

# Project Process discussion

In this section, each phase of the project process will be discussed in terms of description, difficulties (hardware, parts, etc..) and successes. The project process will also be briefly introduced in the following paragraph.

The project process consisted of multiple parts in order to reach the final complete ArduCopter. The project process looks somewhat similar to a spiral model. The first phase consisted of building and understanding Teensy++ applications using low level C code. The second phase consisted of integrating motors with the Teensy++ applications. The third phase consisted of replacing the Teensy++ with a Beagle board, which would now send the PWM signals to the motors. The fourth phase was integrating the ArduCopter with the beagle setup.

The four phases of our process are illustrated below:

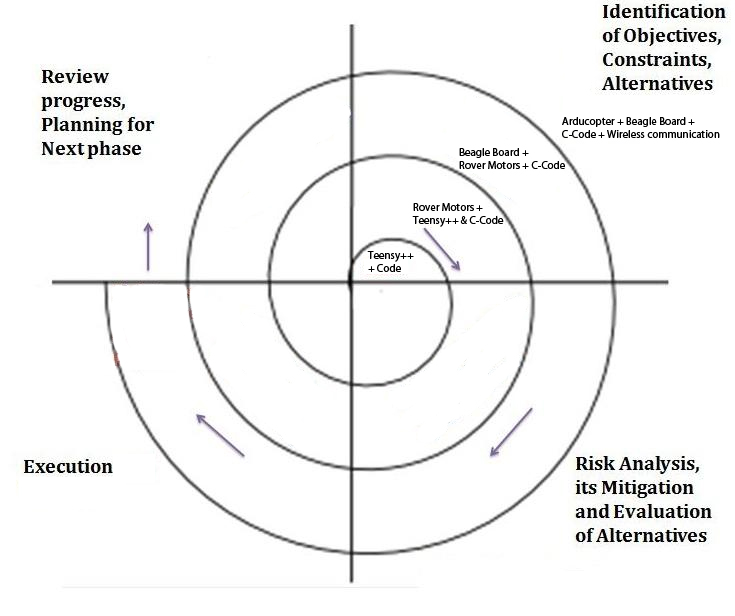


Figure 9: Project Process

## 8.1 First Phase – Teensy++ and C Code

### Description

The initial part of the project was to small system, which covered a Teensy++ and LED’s and C-Code. The purpose of this part was to get familiar with the hardware timers and low-level code.

### Difficulties

The primary difficulties faced were the initial cost of learning the technology. We felt that the instruction steps listed in numerous technical documents contained insufficient effectiveness to convey the idea. We also had to learn the C language because certain members of the team did not have knowledge of the C language. The environment setup in terms of compiler usage and linking libraries was important and was problematic during the development process in this phase. In order to fix the linking process, we built a make file that had the proper configures centralized in one make file and allowed the build process to happen without issue.

### Successes

The successes from this phase was the end result. We managed to learn a lot of new technologies and understand the inner workings of the processor we used on the Teensy++. The initial cost of learning was expensive, but this first phase allowed us to setup our tools for future usage in phases 2 and onward.

## Second Phase – Teensy++, Rover Motors and C Code

### Description

The second phase incorporated the rover motors in conjunction with what we had already built in the first phase. The purpose of the motor inclusion was to get familiar with sending PWM signals to motor-like devices from a Teensy++. The code requires us

### Difficulties

Certain difficulties existed when trying to construct the rover, since the instructions for the rover did not come with the hardware.

### Successes

The motors were easy to understand when adding them to the previous phase setup.

## Third Phase – Beagle Board, Rover Motors and C Code

### Description

The third phase incorporated the beagle board along with the Teensy++ board.

### Difficulties

The primary difficultues with the hardware was with the beagle boards ability to receive output from the ttyACM0. The beagle board would freeze everytime we initiated to read the buffer, which was receving data from Teensy board. This was a confusing problem since it produced similar issues on other beagle boards we got from Raymond Bruton. We assumed this issue could have been caused by configurations of the OS, so we reinstalled the OS multiple times but the problem still persisted. The problem was unresolved.

### Successes

The beagle board was able to sucessfully send PWM signals without any setup issues.

## Fourth Phase – ArduCopter, Beagle Board, C Code and Wireless Communication

### Description

The fourth phase involved setting up the ArduCopter in conjunction with the beagle board and trying to setup the bluetooth, wireless wi-fi and webcam.

### Difficulties

We initially had issues with streaming webcam through the network setup, but we eventually got this to work. The motors on the ArduCopter had problems working properly at the right frequency. We initially assumed this was due to damage by the screws, but we believe this was due to the power supplies in the lab. These power supplies did not provide the power effectively.

### Successes

The ArduCopter on board control system was easy to setup and did not require much testing or setting up. We looked at minor setup information provided online.

## Project Overview

This project allowed us to learn new problem solving techniques with relation to embedded low-level systems. The problems can be coming from infinite possible places, but by certain problem solving techniques, we can now better pin point issues. We also learned a lot about Linux and how to setup certain tools and how to interface with other low level devices, such as using the ttyACM0 to receive inputted buffer data from another device. We also learned about hardware timers and how this hardware timer can quite possibly be used in any possible program to create frequency/time based solutions.

These are valuable aspects because we feel they can be directly applied in the work environment, at companies that have both a software and hardware development aspect.

# Appendix

Include the source code of all programs you developed. Give references to any modules you may have imported from other sources (web, libraries). Include appropriate comments and explanations where appropriate.

## Software Source Code

The software required to run the two major components of code were completed in order to make our project execute, they were the web camera streaming software, the PWM interface in Python. The files below are the code as it stands on the ArduCopter of stuff we have finished that may not be exactly finished.

### ChangePWM.py

In order to send PWM signals over the connected wires to the ArduCopter it was necessary to create a python script that would accept commands and then modify the values as required. Below is the mostly finished script, we simply had to do a bit more tuning in order to make the high/low values for each more accurate. The final version of this code would have included opening the Bluetooth communication server and blocking on a receive call for XBee data. A simple communication format as described would then let us pass PWM values over the network to this script to modify the flight of the machine.

|  |
| --- |
| #!/usr/bin/env python  from bbio import \*  from time import sleep  ROLL = PWM1A  PITCH = PWM1B  THROTTLE = PWM2A  YAW = PWM2B  LIST = [ROLL, PITCH, THROTTLE, YAW]  ARG\_SEP = ","  VAL\_SEP = "-"  def setup():  for x in LIST:  pwmFrequency(x, 500)  #arming sequence  pwmWrite(PITCH, 190)  #sleep(1)  pwmWrite(ROLL, 190)  #sleep(1)  pwmWrite(YAW, 240)  #sleep(1)  pwmWrite(THROTTLE, 140)  sleep(2)  #start sequence  pwmWrite(PITCH, 190)  sleep(1)  pwmWrite(ROLL, 190)  sleep(1)  pwmWrite(YAW, 190)  sleep(1)  pwmWrite(THROTTLE, 190)  def changePWM(pin, val):  if pin == "t":  pwmWrite(THROTTLE, val)  elif pin == "r":  pwmWrite(ROLL, val)  elif pin == "p":  pwmWrite(PITCH, val)  elif pin == "y":  pwmWrite(YAW, val)  def processCmd(line):  line = line.lower()  vals = line.split(ARG\_SEP)  for v in vals:  s1 = v.split(VAL\_SEP)  t = int(s1[1])  print(s1[0] + "--" + str(t))  changePWM(s1[0], t)  def loop():  x = int(raw\_input("Enter a PWM throtle value: "))  processCmd(x)  run(setup, loop) |

### Start\_stream.sh

This simple script has been added to the path of our Copter, following the complete installation of this project to the device and the proper binaries/script being added to our path this was all that was needed to bring up streaming of the webcamera. So long as Wireless is available a user should be able to view the stream at: 1.2.3.4:9000?action=stream, where 1.2.3.4 is the IP that should be fixed (likely via DHCP lease to the Patriot).

|  |
| --- |
| #!/bin/sh  # Use input\_testpicture.so -d 500 -r 320x240 to eliminate source as problem.  mjpg\_streamer -b -i "input\_uvc.so -d /dev/video0 -r 320x240 -f 15" -o "output\_http.so -p 9000 -w ./www" |

### In Progress Bluetooth Code

The following code snippet allows one to sniff all of the Bluetooth devices in the vicinity and get their device information in order to set up a connection. All of these require the properly installed Bluez package as already covered under system setup.

|  |
| --- |
| #!/usr/bin/env python  # Uses the default bluetooth to scan and print all address/names.  import bluetooth  print "Searching for bluetooth devices."  nearby\_devices = bluetooth.discover\_devices(lookup\_names = True)  print "Found %d devices" % len(nearby\_devices)  for addr, name in nearby\_devices:  print "%s - %s" % (addr, name) |

The following two snippets demonstrates a simple connection over Bluetooth using an L2CAP socket (similar to UDP to transmit data). This would have been the preferred method allowing for short well defined packets to be sent over Bluetooth and handed off to the processCmd function in the above ChangePWM.py script. Had we continued further with the XBee and fully proven this to be viable, we would have fully integrated the server portion into the ChangePWM.py script.

Server (receiving):

|  |
| --- |
| #!/usr/bin/env python  # Example server side of socket connection.  import bluetooth  # Required to be odd number in range 0x1001-0x8FFF  PORT = 0x1001  s\_sock = bluetooth.BluetoothSocket(bluetooth.L2CAP)  s\_sock.bind(("", PORT))  s\_sock.listen(1)  c\_sock, address = s\_sock.accept()  print "Accepted connection from ", address  data = c\_sock.recv(64)  print "Received [%s]" % data  c\_sock.close()  s\_sock.close() |

Client (sending):

|  |
| --- |
| #!/usr/bin/env python  # Simple example to connect a datagram bluetooth socket.  import bluetooth  # Required to be odd number in range 0x1001-0x8FFF  PORT = 0x1001  BD\_ADDR = "01:23:45:67:89:AB" # Change this addr to device.  sock = bluetooth.BluetoothSocket(bluetooth.L2CAP)  sock.connect((BD\_ADDR, PORT))  sock.send("Hello")  sock.close() |

# References

|  |  |
| --- | --- |
| [1] | P. Hollands, "DIYDrones," 29 June 2011. [Online]. Available: http://diydrones.com/profiles/blogs/matrixpilot-31-released. [Accessed 10 December 2012]. |
| [2] | "ArduCopter," 10 December 2012. [Online]. Available: http://code.google.com/p/arducopter/. [Accessed 10 December 2012]. |
| [3] | http://www.robotc.net/, "http://www.robotc.net/," 10 December 2012. [Online]. Available: http://www.robotc.net/files/win7\_prolific\_drivers.zip. [Accessed 10 December 2012]. |
| [4] | "Using ICS (Internet Connection Sharing)," Microsoft, 10 Decemember 2012. [Online]. Available: http://windows.microsoft.com/en-US/windows-vista/Using-ICS-Internet-Connection-Sharing). [Accessed 10 December 2012]. |
| [5] | J.-E. Sneddon, "omgubuntu.co.uk," N/A, 31 January 2011. [Online]. Available: http://www.omgubuntu.co.uk/2011/01/realtek-rtl8192su-ubuntu-driver-fix. [Accessed 10 December 2012]. |
| [6] | "BeagleBoardUbuntu," Wiki, 3 December 2012. [Online]. Available: http://elinux.org/BeagleBoardUbuntu#Wi-Fi\_Networking\_.28command\_line.29. [Accessed 10 December 2012]. |
| [7] | A. Hiam, "GITHub," N/A, 10 December 2012. [Online]. Available: https://github.com/alexanderhiam/pyserial. [Accessed 10 December 2012]. |
| [8] | "SourceForge MJPG\_Streamer," N/A, 10 December 2012. [Online]. Available: http://sourceforge.net/projects/mjpg-streamer/develop?source=navbar. [Accessed 10 December 2012]. |