Android Drone:
REMOTE QUADCOPTER CONTROL WITH A PHONE

BY AUBREY JOHN RUSSELL
UNDERGRADUATE OF COMPUTER ENGINEERING

ADVISED BY MR. JEFF GERFEN
PROFESSOR OF COMPUTER ENGINEERING

FALL 2016

12/20/2016

COMPUTER ENGINEERING SENIOR PROJECT

CALIFORNIA POLYTECHNIC STATE UNIVERSITY, SAN LUIS OBISPO

Contents

INTRODUCTION

PROJECT OVERVIEW

The purpose of the "Android Drone" project was to create a quadcopter that can be controlled by user input sent over the phone's Wi-Fi connection or 4G internet connection. Furthermore, the purpose was also to be able to receive live video feedback over the internet connection, thus making the drone an inexpensive option compared to other, equivalent drones that might cost thousands of dollars. Not only that, but the Android phone also has a host of other useful features that could be utilized by the drone: this includes GPS, pathing, picture taking, data storage, networking and TCP/IP, a Java software environment, and a large, diverse variety of Android software endpoints that allow for things like facial recognition or motion detection.

Ultimately, this project was intended to give a cost effective means of adding hundreds of potential features to a standard quadcopter. Although there were other ways of implementing this project, such as buying a Wi-Fi or GPS module and connecting it directly to a microcontroller, the Android phone option represents a much more versatile option that includes significantly higher processing power for the future development of this project.

PROJECT GOALS AND OBJECTIVES

Goals:

- Allow a user to remotely control and fly a quadcopter.
- Allow a user to receive feedback via the phone.
- Create and Android and PC app that allow for future development.

Architecture Objectives:

- Use an Android phone application to use TCP/IP and USB to communicate with the PC program and MSP430 microcontroller respectively.
- Use the MSP430 to control motors with pulse width modulation.
- Use a PC program to take user input and send it to the Android application.

PROJECT OUTCOMES AND DELIVERABLES

Outcomes:

- Future development through the use of Android java endpoints—these endpoints are software packages that include features and functionality.
- Demonstration of Computer Engineering related skills.
- Completion of the senior project requirements.

Deliverables:

- Assembled quad copter kit.
- Working microcontroller code that receives data from the Android phone to control drone motors.
- Functional Android app that can control the microcontroller and receive data wirelessly from a PC.
- A PC java application that remotely controls the drone from any computer.
- A stable, flying quadcopter drone.

FORMAL PRODUCT DEFINITION

OVERVIEW

The “Android Drone” project utilizes a combination of hardware and software that is relevant to a computer engineering senior project. The project includes the Java PC program, the Android java app, the MSP430 Microcontroller, the MSP430 microcontroller C code, the electronic speed control modules (ESC), a 3000 maH Li-po battery, and the three phase AC motors.

The first step of flying the drone occurs on the PC application—the PC first initializes a network connection with the Android phone for two-way communication; the user uses keystrokes to control the quadcopter. In addition, the PC application can send bytes, which the Android phone can receive and forward to the microcontroller or do additional processing. The Android phone, which is connected through USB, sends data to the MSP430 microcontroller, which uses the information it receives to change the pulse width modulation being sent the ESCs. Furthermore, the Android phone sends live video back to the PC over a different TCP/IP connection.

Lastly, the ESCs change the three phase power signal going to the AC motors. The ESCs regulate the three phase AC signal being sent to the drone motors, which has sizeable efficiency advantages compared to a DC motor.

MARKETING REQUIREMENTS

1. The device shall be remotely controlled.
2. The device shall be easily controlled.
3. The device shall be able to send back live video feedback.
4. The device shall be able to fly.
5. The device shall be controlled with keyboard controls.

ENGINEERING REQUIREMENTS

1. The system shall have a PC application that the user interacts with.
   1.1. It shall receive keyboard input.
   1.2. It shall be able to create a TCP server that can be connected to.
   1.3. It should be able to send and receive data with any Android phone it is connected to.
   1.4. It shall be able to reconnect with whatever it is connected to in the event that connection is lost.
2. The system shall have an Android application.
   2.1. It shall have the ability to be a TCP client and connect to a TCP server.
   2.2. It shall have the ability to be a TCP client and connect to a TCP server.
   2.3. It shall be able to have multiple threads that handle sending data, receiving data simultaneously, while also communicating with the microcontroller in real time.
   2.4. It shall have the ability to reconnect with the server if the connection is lost.
   2.5. It shall have the ability to take pictures remotely and store them on the data storage.
   2.6. It shall be able to connect and send data over UART to the microcontroller.
   2.7. It shall revert to an idle sleep mode when not doing anything.
   2.8. It shall have the ability to send back live video feedback.
   2.9. It shall log diagnostic data to the device storage in the event of a crash.
3. The system shall have a microcontroller.
   3.1. It shall be able to receiving data over software UART via a USB cable from the Android application.
   3.2. It will run at a 16Mhz clock and run at 9600 baud.
   3.3. It shall run C code.
   3.4. It shall output four changeable PWM signals at 400Hz that can all run at different widths that rely on the same timer.
   3.5. It shall go into low power mode two when not interrupted.
   3.6. The device shall have an emergency landing function in the event that the USB connection is lost.
4. The system shall use four electronic speed controllers.
   4.1. It shall take in a pulse width modulation signal between 50-450Hz
   4.2. It shall output a three phase AC signal.

5. The system shall utilize four three phase AC motors.
   5.1. Two of these motors shall have clockwise blades and spin in that direction.
   5.2. The other two motors shall have counterclockwise blades and spin in that direction
   5.3. The four motors shall be powerful enough to life the quadcopter and phone.
   5.4. The drone shall take off and hover for at least two minutes.
   5.5. The motors shall be able to move the drone North, West, East, and South.

6. The device shall run on a lithium ion polymer battery
   6.1 It shall use 11.1 volts.
   6.2 It shall be a 30C Li-po battery.
   6.3 It shall be rechargeable and weigh less than 1 pound.

The engineering requirements above will provide direction for the development of the Android drone app. Here are the most important characteristics and attributes of the app with highest priority first.

Stable and Controllable flight:

The drone must be able to achieve stable flight and be able to not only take off, but also land and move in multiple directions.

Flight Duration:

The drone should be able to fly for at least a few minutes and not crash during that time.

Autonomous Control Functions:

The drone programs must have autonomous software functions that automatically do certain tasks—such as takeoff and landing.

User Interface Friendliness:

User friendless is a function of how easy it is for the user to control the drone from the PC. The keystroke input and connection process must be simple and smooth. Once the device is connected the user should only have to connect the two devices over the network and then only have to focus on controlling the drone. Getting video feedback is also nice for user friendliness so they can see where the drone is going.

Fast and Reliable Wireless Control:

The drone must be controllable in real time and therefore the system must prioritize getting the commands to the microcontroller and reconnecting the Wi-Fi if it is disconnected.

Battery Duration:

A lengthy battery lifetime is important for giving the drone the most airtime possible. The Android phone must also last at least as long as the battery lifetime so that the microcontroller doesn't lose connection due to a low phone battery.

Future Development:

The Android app must be written in a maintainable way so that future additions are simple to add. The app needs to be setup so that endpoints can be added without major difficulty.

DESIGN AND JUSTIFICATION

PROCESS OVERVIEW

Designing the Android Controlled Drone was based on a process of identifying the most important functionality that the drone would need to have; the drone was built around the fact that it would need to be able to lift a decent amount of weight. This weight is primarily the Android phone. Furthermore, the drone parts were selected for cost and simplicity due to the fact that the software was going to be a significant challenge and lowering costs was one of the primary goals for this project.

Selecting which hardware would fulfill these requirements was based on creating spreadsheets where important attributes like weight, cost, complexity, and others could be easily compared. The equation below helped to evaluate which hardware should be selected.

$$ComparisonCoefficient = \sum \binom{n}{r} \frac{cost(r)^2 - reviews(r)^3 + complexity(r) + liftforce(r) + familiarity(r)^2}{weight(r)}$$

The higher the comparison coefficient is, the worse the choice is. Dividing coefficients shows how much worse a hardware set is compared to another. The equation sums over the all the individual pieces of hardware in particular hardware sets that were looked at. Although things like cost are self-explanatory, things like complexity are obviously much more difficult to determine, and so those values were mainly chosen through personal judgment. Furthermore, the exponents represent the value of a particular category. Lastly, all attributes are proportional to the weight since anything that is significantly heavier will always produce a lower comparison coefficient.

After plugging in several sets of data into the equation and spreadsheets, while trying different combinations of compatible hardware, the hardware was chosen based on the lowest comparison coefficient. This hardware set contained the drone kit, the microcontroller, the Android phone, the ESCs, the motors, and the batteries. As a side note, the selection for the phone was a personal phone since cost was zero and cost is an important factor.

With the hardware selection completed, the software architecture was also designed from the top down, starting from the highest view—the connections between the drone, the Android phone, and the PC. Next, state diagrams and software flow charts were established for each of the particular component of the system. With respect to the PC application, a system had to be implemented that could handle multiple threads, simultaneously receive keyboard input, transmit to the server, and also receive if need be.

Furthermore, the Android application needed a more advanced software flow diagram because the Android application needed threads for receiving from the server, transmitting to the server, running the GUI, writing serial data to the microcontroller, and taking photos and videos; the threads were essential because the app needed good real time performance. The Android app required a significant amount of planning and development, especially with how to get it to work with the microcontroller—it was not an easy task to plan out how to write serial data. Moreover, the USB software UART was developed by looking at examples online and testing various libraries and drivers.

In addition, the MSP430 microcontroller required a lot of coding because there were challenging performance constraints on the microcontroller; it had to receive data over UART and simultaneously handle four PWM outputs. Therefore, designing how the interrupts would operate was essential for making the microcontroller run as fast as possible. One problem that had to be considered, for example, was the fact that making the UART interrupt service routine too long would interfere with the PWM timer. This would create fluctuations in the PWM output and ultimately make it inconsistent whenever new input from UART was received—the motors would behave unexpectedly and lead to instability. The fluctuations were also verified with an oscilloscope. In effect, any kind of fluctuation in the PWM output could send the drone into a spiral. Overall, a large portion of this development was accomplished through trial and error and experimentation due to a large number of unpredictable problems. The PWM output had to be calibrated for each motor.

SYSTEM ARCHITECTURE

The project architecture can be modeled with a top level diagram that shows how all the components of the architecture are connected. This includes the network, USB, motor, and ESC connections. Because this is the general architecture, it does not cover the details of how the applications specifically operates, although the hardware and software sections cover this thoroughly.

The diagram shown in Figure 1 displays how everything is connected. The PC application can communicate with the Android phone over TCP and vice versa. The Android phone simultaneously communicates with both the PC application in TCP and the MSP430 microcontroller over USB UART for sending and receiving data. Furthermore, the MSP430 microcontroller has one unidirectional communication via PWM with the ESC modules, and those in turn control the motor speed. The ESCs draw power from a lithium ion battery to supply the motors with three phase AC power. Moreover, in order to control the drone motors from the PC application, a data packet must traverse over WI-FI to the Android phone, then passed to USB UART, and then finally sent as a modification of the PWM signal for a particular motor. This setup requires that a WI-FI network already exist and implies that data, such as a live video signal, can traverse back over the TCP connection to the PC.

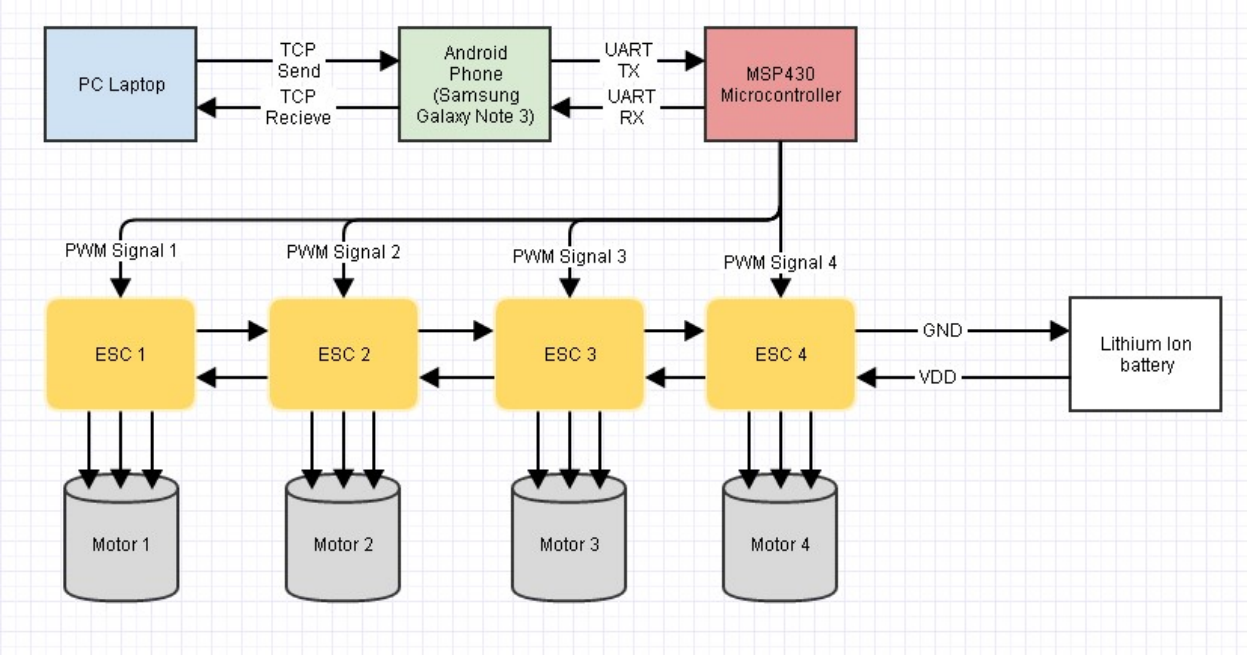

Figure 1: High Level connectivity flow chart

The general system architecture is fairly comprehensible, but relies on significantly more complicated software. The software on the Android phone abstracts most of the complexity away.

The hardware diagrams for the project includes a diagram of the MSP430's connects as well as the layout for the ESCs. In Figure 2 below, the hardware connections and pins are labelled specifically

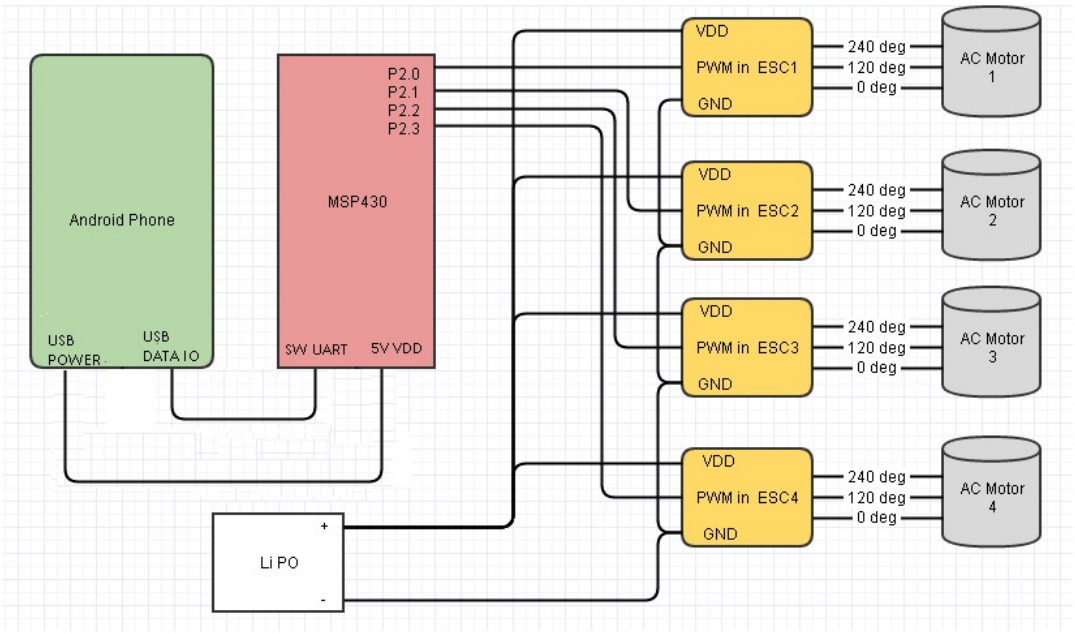

Figure 2: Hardware pin connections diagram

The hardware pin diagram resembles the system architecture diagram, but shows the specific pin connections between the MSP430, the Android phone, the ESCs, and the three phase AC motors. The ESCs shown on the pin diagram, however, are much more complicated and encapsulate a significant amount of logic as shown in Figure 3. The key purpose of the ESCs, as was indicated by research, is to convert the PWM signal, in an efficient way, to a three phase power output that can also protect against current spikes and regulate voltages in a safe way; in this case, safety refers to protecting the drone against crashing in flight and thus protecting people. Rapid changes in voltages could cause the drone to lose stability. The spikes could be caused by changes in battery temperature or a fast change in motor speeds. Furthermore, the AC motors spin dangerously fast, and so protecting against spikes, voltage changes, and fluctuations is important for safety purposes, especially if the drone is in flight.

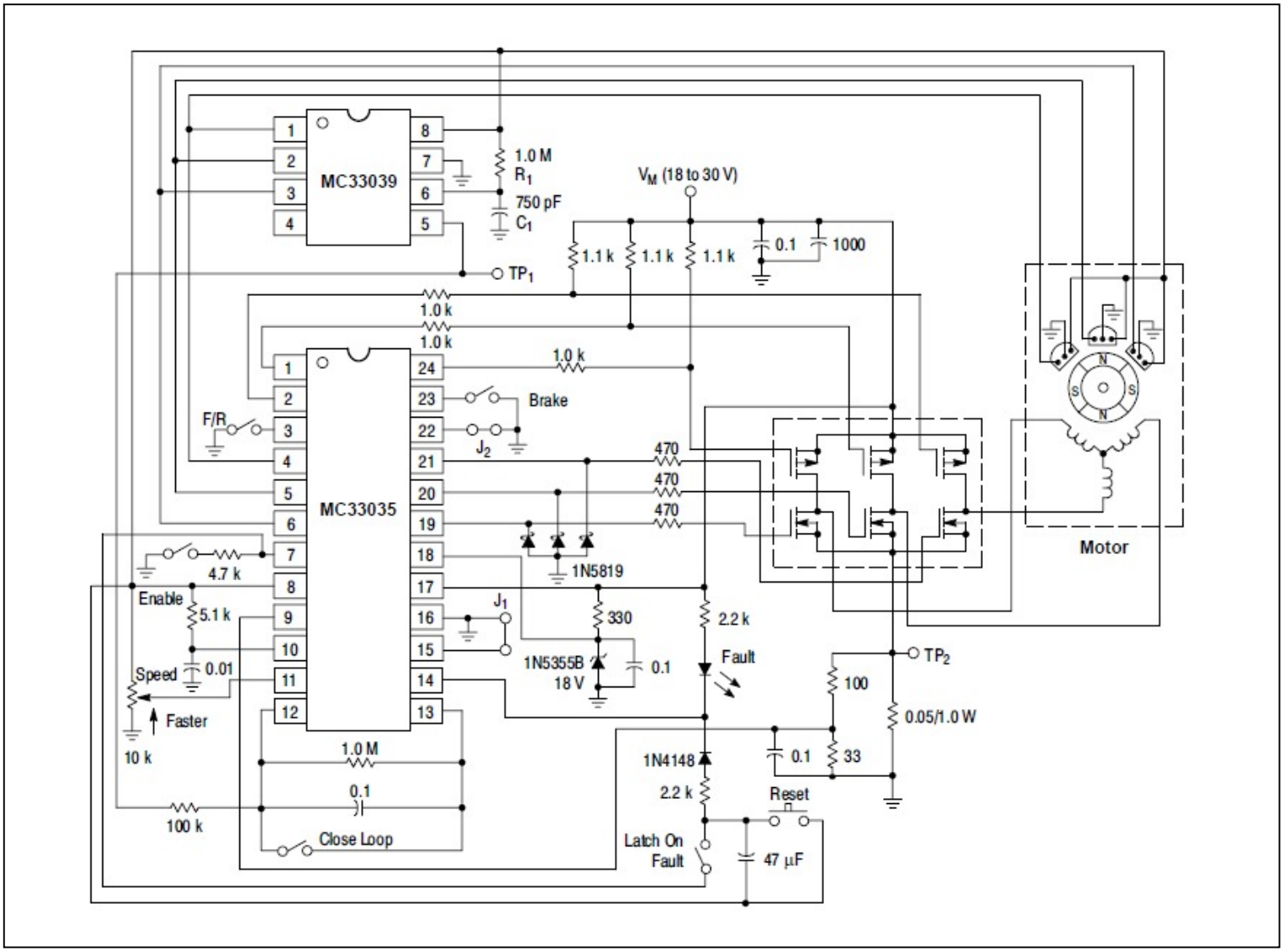

Figure 3: ESC schematic

MSP430:

The MSP430 uses a 5V input and supplies a PWM output on pin 2.0, 2.1, 2.2, 2.3 at 3.7V. In addition, the MSP430 runs at its fastest speed of 16 MHz to improve control of the ESCs. Lastly, it communicates at 9600 baud over software UART back to the Android phone. The MSP430 runs on a Launchpad and is supplied 5V over a USB input. Lastly, it utilizes 4mA of current during maximum draw, at 16 MHz clock speed, and 230 uA in standby mode.

PC Laptop:

The windows PC laptop runs the PC program and hosts the server for the Android application to connect to. The application is written in Java and runs in an environment with a Java Virtual Machine. The Java code takes advantage of a Wi-Fi connection library that is built into Java. Furthermore, the PC laptop has a 2.5 GHz core i5 CPU with 4 GB of ram and a 128 GB solid state drive with an Intel Centrino Advanced WI-FI chip. Finally, the laptop acts as a WI-FI hotspot that the Android phone can connect to.

Android Phone:

The Android phone is a Samsung Galaxy Note 3 and runs on Android 4.4.2. It has a 1080p video camera, 4G LTE data connection, built in WIFI, a quad core CPU, 4GB of ram, USB peripherals, and a high resolution camera. It also supports Android Java and is completely customizable.

Electronic Speed Controller:

The ESCs take in an 11.1V power input and up to 3A of current. They receive a PWM signal from anywhere between 50-450Hz between 5 and 3V. The ESC creates a three phase power output that changes depending on the PWM. The PWM inputs function between 45% and 70%, where 45% represents the lowest standby mode, and 70% represents maximum power. This gives a range of 25 different levels that the microcontroller can communicate to the ESCs. The ESCs have a built in voltage regulator that holds the voltage above 10V and can supply a maximum total current of 15A. The ESCs will control the motors at a rate of 920 RPM/V. Lastly, the ESCs create various tones to indicate the status of the ESC including whether it detects and input and whether there is a problem or abnormality.

AC Motors:

There are four brushless AC motors that are connected to the ESCs. They can spin at several hundred rotations per minute, at a top speed of 3500 RPM, and all four can lift a total of 5 to 6 pounds at maximum output. Two of the motors spin clockwise and the others spin counter clockwise in order to cancel out the torque created by spinning blades.

There were a variety of hardware options for this project that could have been used. The real considerations were whether to buy custom components or buy a quad copter kit; the quadcopter kit was selected. Although the custom components were less expensive than the F450 quadcopter kit, it wasn't guaranteed that all the individual components would work together; there may have been some incompatibility problems. Furthermore, the initial plan didn't include the use of ESC modules at all—the original plan only included BJTs connected to the PWM signal to control DC motors since the transistors are simpler, lighter, and cheaper. However, research revealed that the AC power motors are significantly more efficient, and one transistor would not be able to produce the required AC output from a DC lithium ion battery.

In addition, the ESCs regulate voltage and help to keep the motors running more consistently despite the voltage drop as the batteries become more and more depleted. The other important consideration was whether using a PWM signal hooked up to a BJT would actually work; every source online strongly recommended using an ESC, and never had any examples of anyone using transistors instead of BJTs. Therefore, the best decision was to utilize the ESC modules.

The other consideration was on which particular phone would be the best choice. Because of cost constraints, the Samsung Galaxy Note 3 owned by Aubrey Russell was the ideal selection, since didn't cost any extra money, it utilizes a modern version of Android, it has a relatively good camera, and it has WI-FI; in order to protect the phone against the damage from a potential crash, the mechanical design included some Styrofoam padding attached to the bottom of the drone, which was sufficient to absorb a good amount of shock from a fall of five feet.

Although the hardware components selected worked relatively well for a while, the ESCs in particular proved to be somewhat inconsistent and unreliable. The first set of ESCs were working effectively and able to control the four motors equally such that the quad copter could hover and fly as intended. These motors could be calibrated by attaching the drone to a rope—the rope prevented the drone from moving too much so that it would be possible to narrow in on the values needed to create an equal amount of lifting force on each motor.

However, when doing further development, two of the ESCs stopped working. One ESC had a short and glowed red hot until the power was disconnected, which also destroyed one of the batteries. It appears that one of the capacitors melted, and the proceeding short drained the battery below its minimum operating voltage—going below this voltage breaks the battery and prevents it from charging again. The second ESC simply stopped working and would no longer take an input. The next step was the replacement of these ESCs, but the problem was that the next two ESCs that were installed were very different from the other two from the first set of ESCs.

It was difficult and time consuming to calibrate the motors based on these new ESCs. The drone had to be tied down and the motors had to be adjusted very slowly so that the drone would be stable; this was made especially difficult because the ESCs and motors require different PWM signals to achieve the same rotations per minute and therefore the same lift. For example, an increase in PWM for one ESC didn't increase the rotations per minute of the other motors by the same amount in. So an increase in 1% for one motor was equivalent to an increase in 1.45% for another motor, for instance. This made it almost impossible to calibrate because floating point operations of the MSP430 are inviable for a real time system. Lastly, without a higher MSP430 clock speed, the PWM signal could not be increased by smaller increments and 1% was the best possible increment. Despite problems with the second set of ESCs, the drone did work for a while and would work if there was a similar set of ESCs.

The software for the Android Drone was developed from the bottom up by first getting the PC program and Android application communicating and exchanging data. Figure 4 shows the software flow diagram for the PC program and Android application as well as how the Android app sends data over USB to the microcontroller. The first step for both the Android app and PC program is the initialization of the GUI thread; the GUI thread allows the user to interact with the Android app and the PC program even when the Android app and PC program are transmitting or receiving data. Moreover, this also includes the event listener thread that triggers once a key is pressed. However, if the Android app or the PC did not have its own GUI thread, then there would be poor performance anytime there was an exchange of data—something that is not ideal when real time performance is necessary. The other parts of Figure 4 are the threads for the TCP communication.

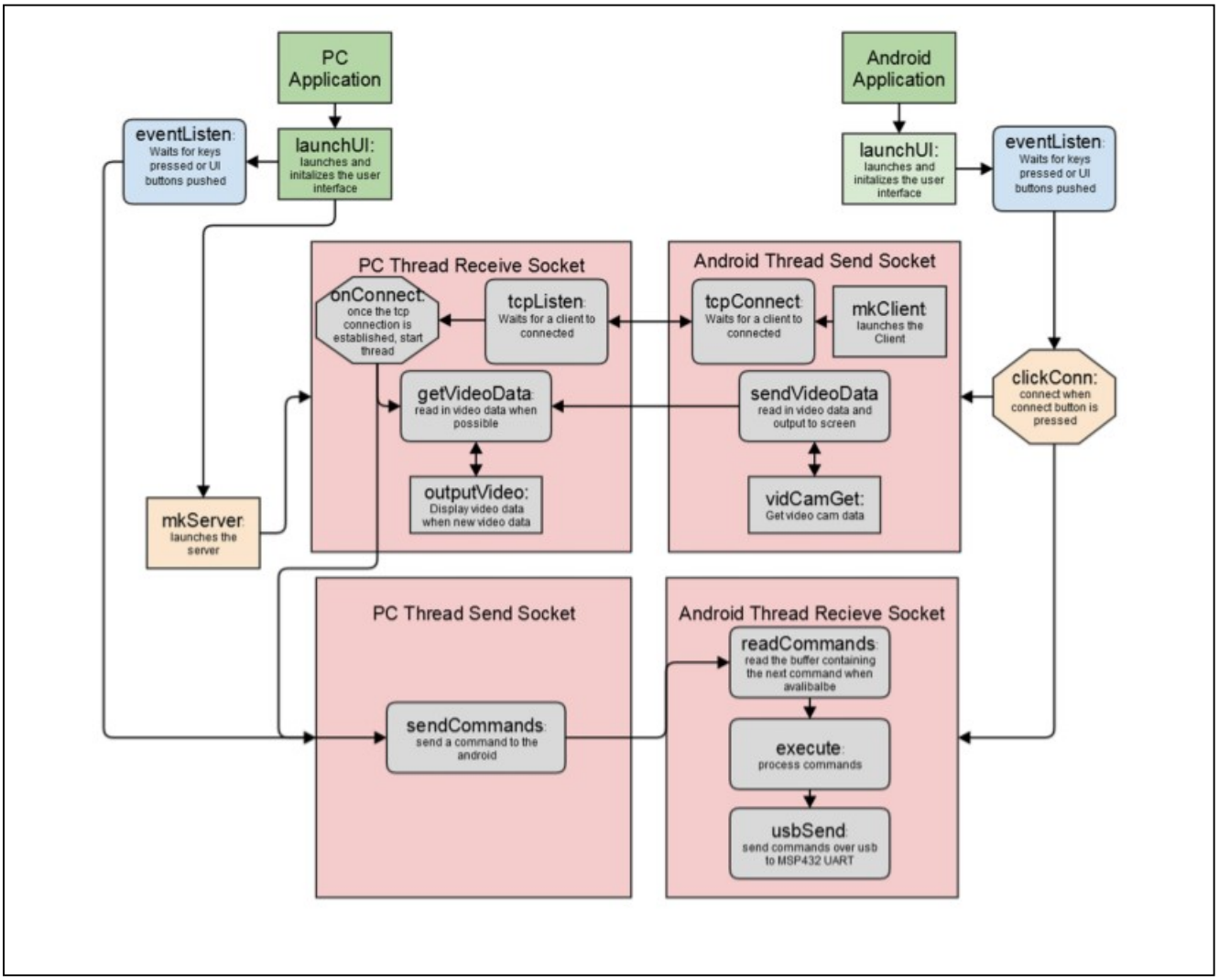

Figure 4: Software flow diagram between Android app and PC program.

After setting up the GUI thread, the PC program and Android app begin to establish the TCP send and receive threads. The send and receive threads allow the exchange of data and automatically recover in the event that the connection is broken. Furthermore, another thread is created on the Android app that writes a serial output to the USB in the event that the TCP receive thread gets some data from the server.

So, for example, if the PC detects the keystroke to increase the speed of the motors, the TCP server will send data to the TCP client over WIFI. The client parses the incoming data and creates a thread handler that transfers the data to the serial UART writing thread, which is sent to the MSP430. Once the data is received by the MSP430, it changes the PWM values as shown in Figure 5. Then, once the timer triggers, decides whether or not to create a high or low value on a respective pin. This ultimately changes the output to the ESCs.

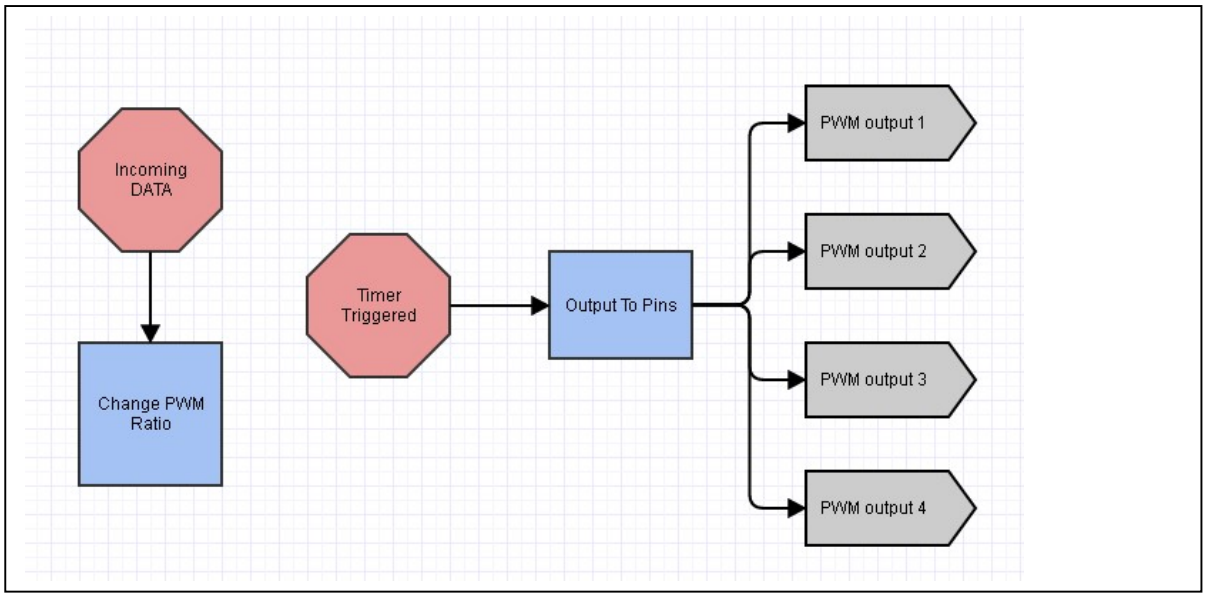

Figure 5: Software Architecture of MSP430 microcontroller

PC Program:

The PC program sets up the user interface and also handles the communication with the Android app; the PC program includes up a keystroke listener that passes data to the TCP network connection. The PC program can also receive any data—whether that be diagnostic log data or pictures or a live video feed. However, the PC program also leaves an opening for a different method of controlling the drone—instead of listening to keystrokes, the PC program could use a joystick library or an Xbox controller library that would enable an easier control method compared to keyboard commands. Lastly, the PC program can be programmed to run automatic macros—such as pressing a key the slowly lowers the drone motor speed to allow it to automatically land.

Android Application:

This is one of the more complicated parts of the project. The app receives data, parses it, and forwards it to the MSP430 over a USB serial UART connection. The app essentially acts as the interface between the user endpoint and the hardware and offers a significant amount of future development potential. Other Android apps or Java libraries can be taken advantage of that would enable things like facial recognition or AI pathing or route planning by using google maps; the vast amount of processing power on Android phones means that Android phones are going to offer much more potential functionality compared to a drone that only has a microcontroller like an Arduino or MSP430. Also, the Android app is generalizable meaning that it can be put on numerous Android phones over version 3.0.0 or be used on a variety of different network connections such as 3G, or 2G, or Bluetooth, or anything else.

MSP430 C Code:

The MSP430 ended up being an excellent choice for this project because of the ability to produce PWM signals with a low cost and low weight microcontroller. Furthermore, the software UART of the MSP430 made it possible to interface with the Android phone, albeit with some difficulty. However, the most important reason for using the MSP430 is the familiarity with this microcontroller that comes from experiences in CPE-329.

PC program:

State diagrams and behavioral models were essential for writing functionally correct and efficient programs that operate as they expect. The first state diagram is the PC program. The program starts in main and creates a new GUI thread where the user interacts. It gets the current IP address and pushes it to the UI; the user can then see it and enter it into the appropriate Android phone input field. Then, it starts the TCP thread, which waits for a connection from the client. When the client connection occurs, the PC and Android application setup a new connection where two-way communication can occur. With the connection established, the PC program waits for either a user key stroke or for client data. If it receives client data, it writes that data to the UI and if the PC program detects a key stroke, it sends the data to the Android application client.

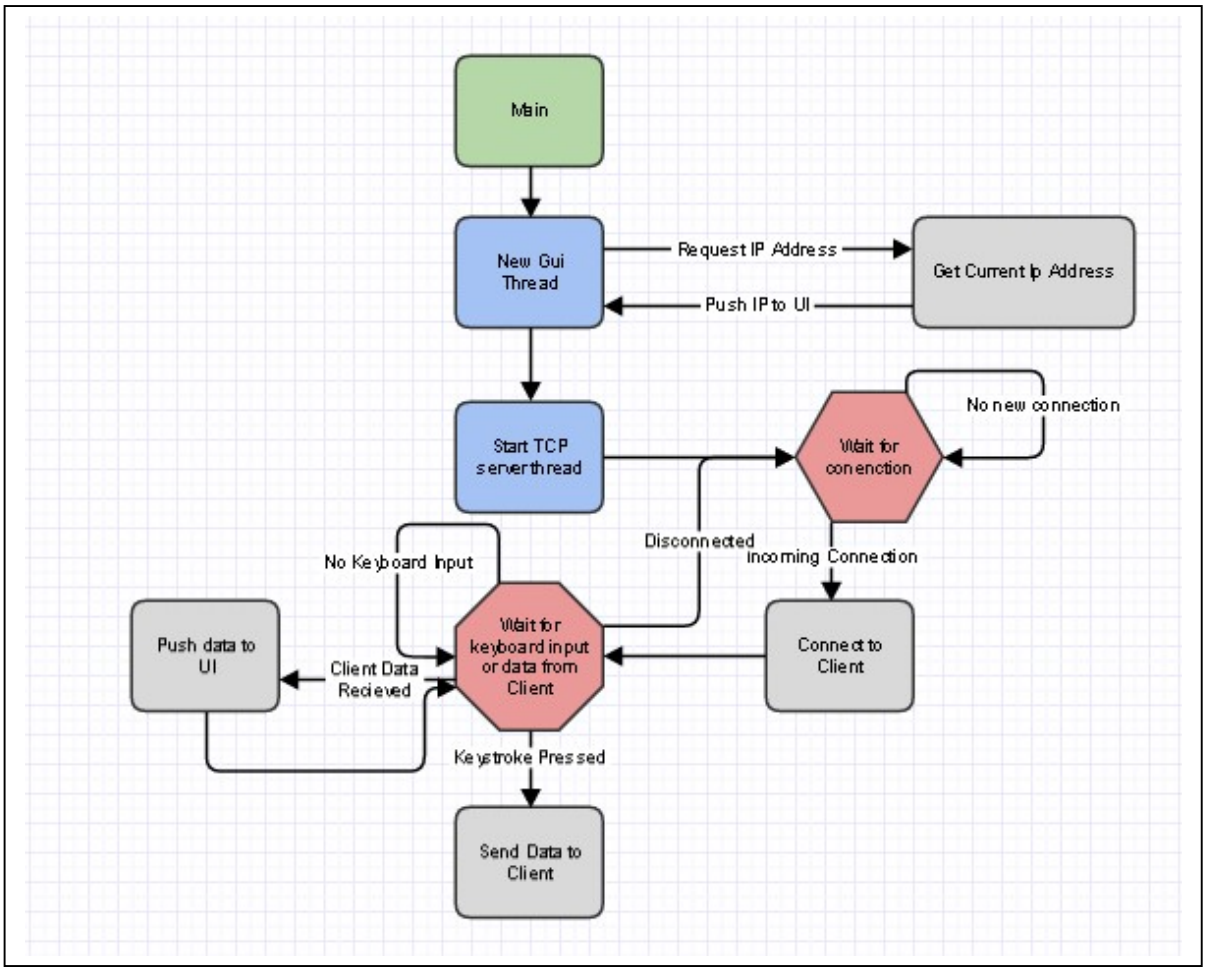

Figure 6: PC program state diagram

Android Application:

The Android app state diagram somewhat resembles the PC program state diagram but ultimately requires many more steps. The major difference is that the Android phone has to acts as the client while also being able to write to the USB output. Like the PC program, the Android application creates a new GUI thread and then proceeds to start a TCP client thread. To get started, the user must enter the IP address and port number, and then press connect. Once the server handshake is complete, client waits for server data or an error. If the client receives a certain command, it might write to USB output so that the output increases or decreases the PWM. Lastly, if the client disconnects, then the client tries to reconnect to the server until it is able to reconnect.

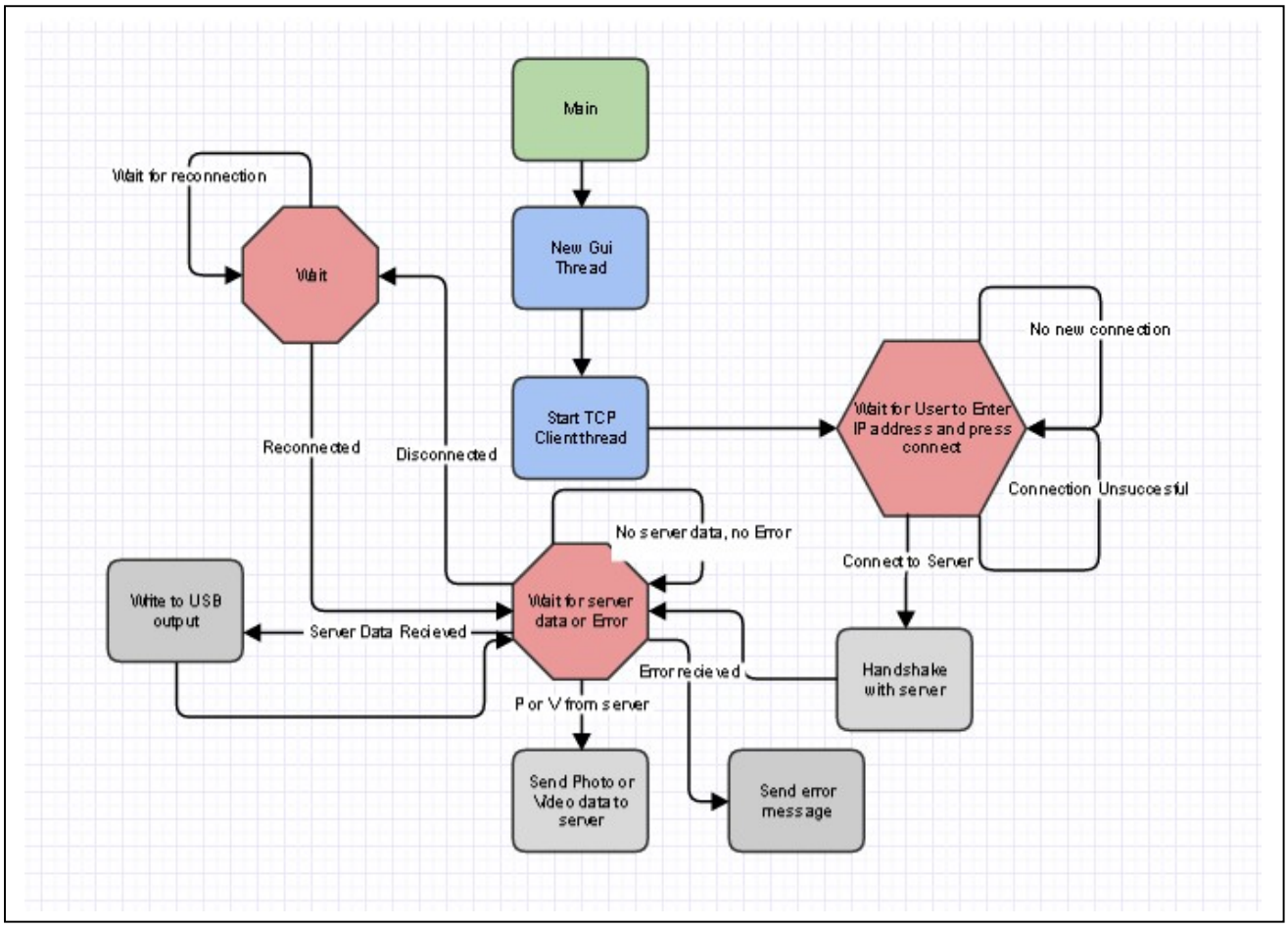

Figure 7: Android Application state diagram

MSP430 C Code:

The MSP430 initializes the 16 Mhz clock, starts Timer A, and starts SW UART at 9600 baud. It also keeps track of several variables like the default PWM settings and the size of the CCR0, and the current counter. Once initialized, the MSP430 goes into a low power and interrupt enabled mode.

Whenever data is received, the UART RX interrupt is triggered and changes the PWM variables respectively. Then, the UART interrupt calls a function called “changePWM”, which is a function that looks at what data is coming over UART and adjusts the PWM accordingly. At the same time, a timer with a counter is running. The counter counts between zero and 100 and increments each time the timer triggers. Each time the counter increments, a function checks if the counter is greater than or less than the respective PWM value during P2OUT set. If the counter is less than the PWM value, the pin output becomes high, and if the counter is greater than or equal to the PWM value, then the pin output becomes a low. This enables all the pins to be changed simultaneously with different PWM values depending on what each motor PWM has been set to. Once “changePWM()” finishes or “P2OUTSet()” finishes, the MSP430 returns back to the idle, low power status. The last case is that some kind of MSP430 error occurs or some diagnostic data needs to be written back to the Android application; in that case, the device enters the TX UART interrupt, where it waits for the TX buffer to become ready and then sends the next character back over UART. Lastly, the MSP430 returns to low power mode when sending data is complete.

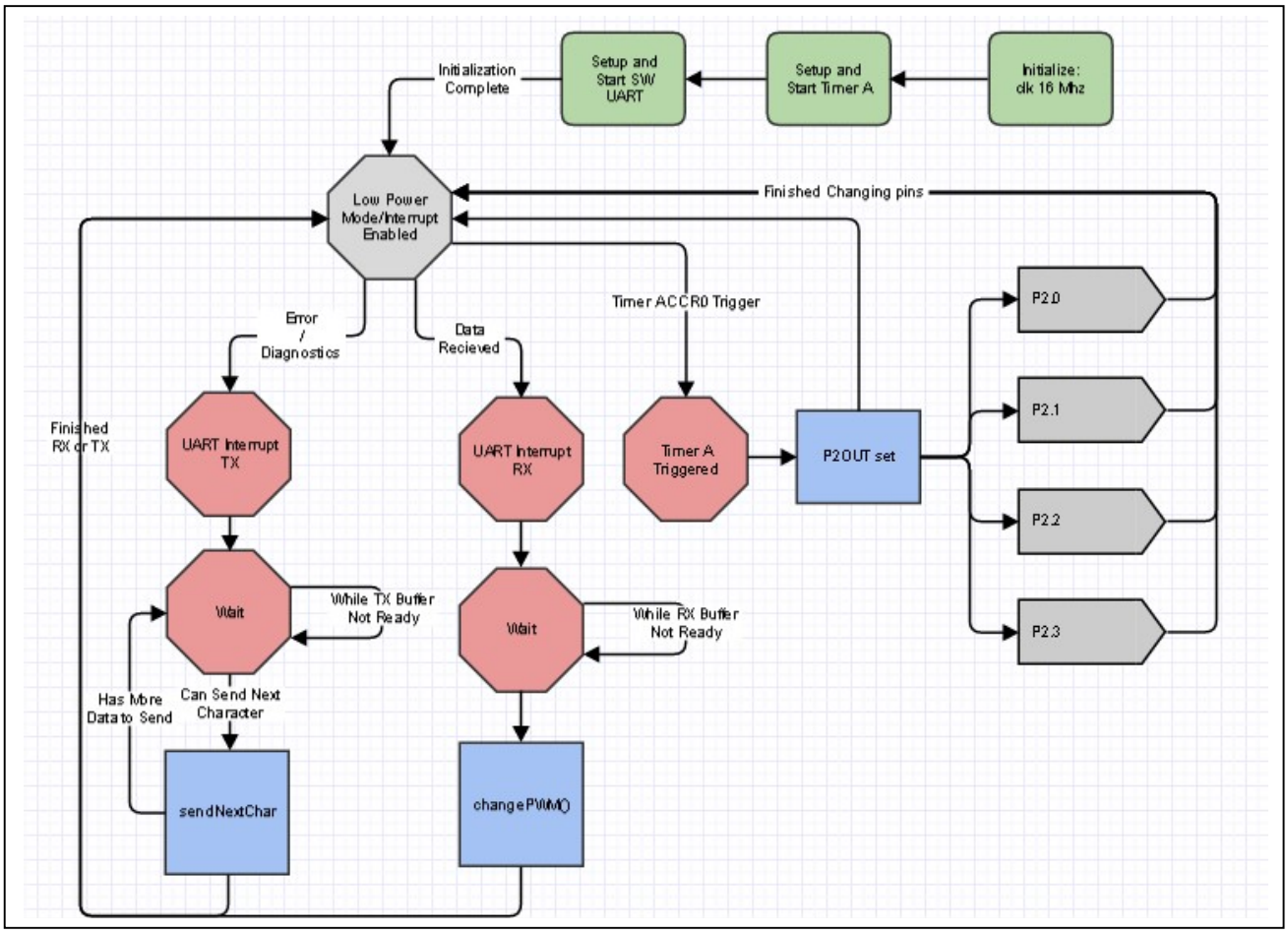

Figure 8: MSP430 state diagram

SOFTWARE DESIGN DECISIONS

The main set of software decisions for this project had to do with the Android application and PC program. One of the most important considerations was determining if the PC program should be the client or the server. Although putting the server on the Android phone and then connecting it with the laptop would have made testing easier—so that the user wouldn't have to constantly connect the phone and then re attach it to the drone—the most logic course of action was to put the server on the PC. This is because the Android phone could easily change IP addresses when flying causes switching between different WI-FI routers or 4G cell towers; this would require a re-initialization of the server and a new TCP connection. Furthermore, the drone would have to be retrieved and would require a user to manually enter the new IP address into the drone in order to reestablish the TCP connection. Therefore, it made more sense to put the server on the PC and have the drone automatically reconnect to that constant IP address whenever it changed IP addresses or otherwise lost connections.

Another difficult software decision was how to design the different programs in the first place. For example, in the Android app, there were considerations about whether to write one consecutive program that handles everything in one thread or tries to create multiple threads to take advantage of parallel processing for performance benefits. The main disadvantage with multiple threads is an increase in complexity and therefore a higher probability of introducing more bugs and race conditions. Race conditions might occur because the order of operations occurring in the threads might affect the state and output of either the PC program or Android application. This could be catastrophic, for example, if the Android Application were to output serial data to the MSP430 before it actually processed the incoming data from the PC application; at best this would do nothing or cause an error, but at worst garbage data could be sent to the MSP430, which would cause anomalous behavior. Fortunately, rigorous state diagrams helped to avoid these problems; the threads were developed such that the serial write only occurs once the data has been fully received from the PC application. The reason for going with multiple threads though was because the Android drone needed excellent real time performance. Taking advantage of the multiple cores of the Android phone was essential to meet those performance demands.

EVALUATION OF SOFTWARE DESIGN CONCEPTS

The software decisions that were made regarding the top down design, in addition to writing code based on the state diagrams in this paper, proved to be an effective strategy. Furthermore, the automatic network reconnection, in the event that the network connection was broken, worked as expected according to the testing data. The phone was always able to reconnect back to the PC program as long as it found a Wi-Fi network or 4G connection.

Also, the software decisions regarding the MSP430 were acceptable. The MSP430 was able to receive data over UART and then control the PWM output for four different signals without too much of a margin of error. The original code for the MSP430, however, did require extensive modification. The CPU clock had to be changed from 1 MHz to 16 MHz in order to achieve the necessary speeds for controlling the PWM signals. This meant that the UART pre-scaler had to be changed to achieve the same UART frequency as before at 1 MHz. Furthermore, the timer CCR0 had to be increased to match the increase of the frequency. In addition, although there are other methods for controlling the PWM, the method that was selected created an easy way of changing four different PWM signals with the use of only one timer. This meant that the program was very efficient since it didn't have to rely on four different timers. The program was also designed with testing in mind—predefined constants in the code allow for variables related to frequency to be changed with only a couple of constants. This accelerated the testing period.

Ultimately, the decisions for the software ended up being relatively successful. The PC program and Android application operate within all specified parameters in a reliable and efficient way. The software components of this senior project are an effective implementation that enables the future development of new features. However, even though the software ended up working as expected, it was not easy to develop and took a large amount of time as well as hundreds of lines of code. With more time there could be some simplification and optimization as well as the inclusion of new functionality.

The mechanical design for the senior project was based on the F450 quadcopter kit, which came with the following: a quadcopter chassis, the necessary screws and bolts, the motor blades, the "blade to motor" attachments, solder-able connectors to the ESCs, and the motors themselves. The solder-able connectors and "blade to motor" attachments made it easier to replace damaged components. Furthermore, the exposed connectors were heat-shrunk and firmly held down to the chassis with zip ties to prevent a short—the short could have caused a toxic and dangerous battery explosion or fire. Furthermore, the battery was not just soldered onto the voltage terminals of the drone. Wires were soldered to the positive and negative terminals on the drone chassis on one end. The other ends of the wires were soldered to a plug, which could connect to the battery. The plug could then be disconnected and reconnected from the battery, which made it possible to manually disable the power source in case of an emergency.

Next, the battery was attached to the drone with several sticky pads and Velcro straps. The final piece of the mechanical design was the attachment of the Android phone. The Android phone stayed on the bottom-middle of the drone and was surrounded with shock absorbing Styrofoam to protect against crashes. This meant that the center of gravity was as low and central as possible. The Velcro straps held the Styrofoam and phone to the bottom of the drone and a small hole was cut in the bottom of the Styrofoam to enable the camera to function without being blocked. Lastly, the MSP430 microcontroller was placed on top of the drone and held down with zip ties to give easy access to the pins. Figure 9 is a picture of the final mechanical design of the drone.

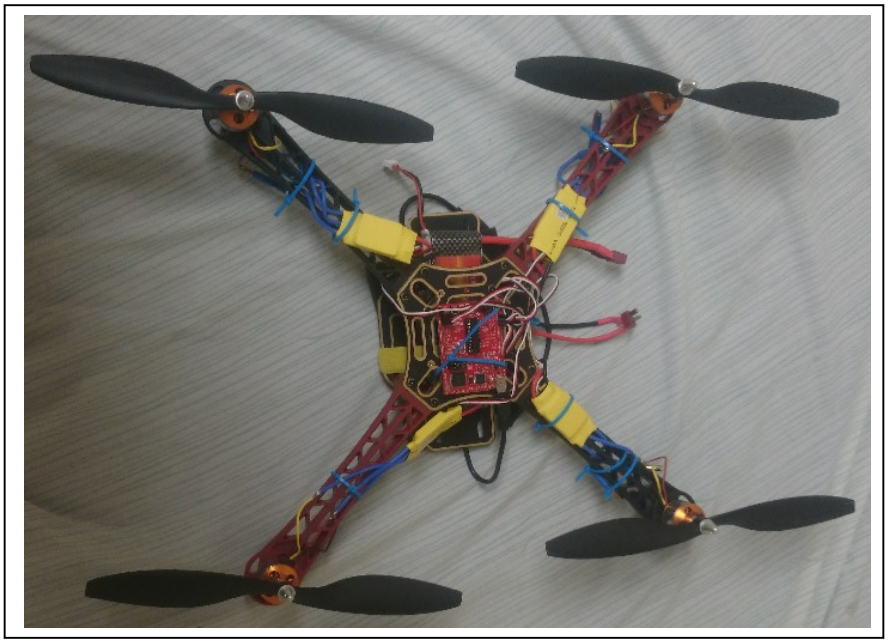

Figure 8: Final picture of Drone

MECHANICAL DESIGN DECISIONS

The design decision of choosing the quadcopter kit had several advantages and disadvantages. The various parts of the kit fit together very well, but the ESCs the kit came with were poor quality and inconsistent. However, everything was compatible mechanically and the drone chassis was not only sturdy, but was also light weight. Furthermore, the propellers, although somewhat durable, would basically break if they hit anything at high velocity. Consequentially, at least 3 propellers needed to be replaced over the course of the project because the blades hit something while rotating rapidly. A major improvement would be some kind of circular guard to protect the blades from material coming from the sides. Ideally, it also would have been nice to create a 3D printed chassis for the microcontroller for aesthetic, structural integrity, and aerodynamics purposes.

SYSTEM INTEGRATION AND TESTING

TEST PLAN—WI-FI reconnect

Experiment	Procedure	Pass Condition
1	Moved out and in of WI-FI range	PASS
2	Moved out and in of WI-FI range	PASS
3	Moved out and in of WI-FI range	PASS
4	Connected to different WI-FI	FAIL
5	Connected to different WI-FI	PASS
6	Connected to different WI-FI	PASS
7	Connected to different WI-FI	PASS

TEST PLAN—MOTOR operation

Requirements	Procedure	Pass Condition
1	Drive Motor 1 to FULL power	PASS
2	Drive Motor 2 to FULL power	PASS
3	Drive Motor 3 to FULL power	PASS
4	Drive Motor 4 to FULL power	PASS
5	Increase motor 1 and motor 2 speed equally	PASS
6	Increase motor 1 and motor 3 speed equally	FAIL
7	Increase motor 1 and motor 4 speed equally	FAIL

TEST PLAN—Keyboard input

Requirements	Procedure	Pass Condition
1	Press “T” Increase all motor speed	PASS
2	Press “T” Increase all motor speed	PASS
3	Press “G” decrease all motor speed	PASS
4	Press “G” decrease all motor speed	PASS
5	Press “R” emergency reset to motor speed 0	PASS
6	Press “R” emergency reset to motor speed 0	PASS

CONCLUSIONS

This senior project was a satisfying and valuable learning experience. Utilizing many foundational computer engineering topics such as interfacing high level languages with low level microcontrollers and controlling PWM driven modules was consistent with Cal Poly's philosophy of "Learn by Doing". The project provided insights into TCP/IP and networking, parallel programming, soldering, and research and analysis. Overall, this project was mostly successful—the drone was able to operate correctly and achieve stable flight until two of the ESCs broke, which made the drone motors impossibly difficult to re-calibrate.

With the two ESCs broken, the drone could still take off, but once it was in flight it would quickly become unstable since the ESCs did not control the motors equally for each change in the PWM signal. In order to make the drone work properly in the future, a faster CPU is needed that can produce smaller PWM outputs in order to increase the motors speed by smaller increments. A microcontroller with at least quadruple the processing power would be ideal to make the increments 0.25% instead of just 1% in the PWM signals. In addition, a simple fix is to use higher quality ESCs that can be calibrated to all perform identically—with identical ESCs the drone would have been precisely controllable. Next, another potential option is including a control system that can determine the orientation of the drone and adjust the motor output such that the drone can automatically adjust itself if it becomes unstable. Lastly, a gyroscope could help to achieve stability.

Despite the setback of broken ESCs, the Android application and PC program both represent aspects of a real time operating system. This real time operating system functioned correctly and was the part of the project that required the most computer engineering skills. Therefore, even though the two replacement ESCs prevented stable flight at the end, the important computer engineering goals were achieved, which makes this senior project a success.

Ultimately, the skills practiced and developed during this project can be utilized in industry where real time operating system constraints and interfacing diverse computer hardware is commonplace. Lastly, this project established the ground work for future development; this project can be expanded by adding new features via the Android phone and its corresponding Java libraries.

REFERENCES

1. F450 Quadcopter User Manual and Datasheet." *DJI Innovations* (n.d.): n. pag. Web. http://download.dji-innovations.com/downloads/flamewheel/en/F450_User_Manual_v2.1_en.pdf
2. Gerfen, Jeff *California Polytechnic University San Luis Obispo*, Professor
3. Texas Instruments. "Mixed-Signal Systems." *MSP430 Microcontroller Basics* (2008): Web. http://www.ti.com/lit/ds/symlink/msp430g2453.pdf
4. "Amazon." *Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & More*. N.p., n.d. Web. 19 Dec. 2016. <http://www.amazon.com/>.
5. "Family User Guide." *Texas Instruments Incorporated* (n.d.): n. pag. Web. 19 Dec. 2016. http://www.ti.com/lit/ug/slau144j/slau144j.pdf
6. *Floureon Batteries*. N.p., n.d. Web. 19 Dec. 2016. <http://www.floureon.com/batteries-e_2/>.

Table 1: Bill of materials for the Android drone

Part	Name	Description	Qty	Unit Price
Phone	Galaxy Note 3	Android phone	1	0
Kit	F450 Kit	Drone Kit	2	78.5
Microcontroller	MSP430	For controlling ESCs	1	18.9
Battery	Fluoreon 2 Pack	11.1V 30C batteries	1	38
Charger	CoolRC B3	Charge batteries	1	13
				Total: $227