# EE 478 Final Lab Report:
# Quadcopter

**Lab Performed By:**
**Rebecca Chu**
**Phongsakorn Liewsrisuk**
**Jian Ma**

Contribution
(Oat) 0xFFFF hours |
(Jian) 0xFFFF hours |
(Rebecca) 0xFFFF hours |

**Special Thanks To:**

Mike for being the quadcopter and breaking our heart every time we fly it
Professor James Peckol for his guidance throughout the project and the "drill more holes advice", and the very delicious pizza.
Ky To for her emotional support and membership fees and eating our snack supplies
Jeremy Zhang in the AA department for help us with the CAD drawings
Professor Blake Hannaford for his advice in PID controllers.
Alvin Chia for supplying us parts from his black market.
Alex Ching for letting us borrow his compass as a replacement.
Our beloved family for supporting us throughout the project
and paying for our overpriced, ripped-off tuition.
Debbie Chu as our late night chauffeur and on call food deliverer
and the makerbot for its dirt-cheap labor for printing the quadcopter leg and camera holder as well as our freedom

# Table of Contents

## ABSTRACT

Our goal of the EE 478 capstone project is to create a quadcopter that serves as a personal camera assistant that will be able to fly up to certain altitude to take pictures for users. This report details our design of quadcopter's flight controller, sensor reading for stability, and photo taking functions. The result of anticipated quadcopter is mediocre. Although the quadcopter did perform some flying maneuvers, the stability still has great room for improvement. Unfortunately, we did not have the opportunity to further develop it due to the consecutive failures of sensor board mounted on the quadcopter. Despite the misfortune, the quadcopter is still able to "fly" (lift off the ground) with no internal stability control.

## INTRODUCTION

The fact that numerous tourists spent the majority of their traveling time taking pictures and recording videos upset us. Believing that human being should be free from such tedious labor, we decided to provide a mean to automate photographic experience using existing robotic technology. After intensive research, we decided to introduce the concept of personal photographer quadcopter. A personal photographer quadcopter is a quadcopter that is able to automatically, or at least intelligently operate cameras for its user from high above. Ideally, we expect the quadcopter to track its user, fly around him or her, being able to find the best camera angle and/or camera setting through a sequence of optimization algorithms. By doing so, it would become a useful tool for traveling, documentation, and recording extreme sports such as snowboarding. However, due to the limitation in resources and time, we restrict our project to mainly building a quadcopter that is able to hover, take 360 degree pictures/videos for at least 10 minutes, and provide server access to stream the data through Wi-Fi.

The first phase of the project, or the main phase, was designing a partially functional first-person-view quadcopter prototype from scratch. Electronic components outsourced include an on-board PC (raspberry pi), 4 motor control boards needed to run brushless motors, and a raspberry pi camera. Sensors we chose include 4 sonar range finders, an IMU with a gyroscope, an accelerometer and a digital compass. Anything else was built from IC level. For the quadcopter frame and protection, we designed them in CAD tools and used wood, aluminum rods for implementation. We designed and soldered a power board(LDOs and power protection), a flight controller board with 1 PIC18 to process data in order to calculate speed output for 4 propellers, a sensor board to extract data from the IMU and fuse data from sensors. In terms of communication, we adapted the Wi-Fi interface in raspberry pi to minimize RF design components due to our limited expertise in that particular area. Aside from standard internet protocols for video streaming, real-time flight control was implemented using SSH.

## DESIGN SPECIFICATION

The quadcopter cameraman system is a system that provides the automatic photographic and videotaping solutions for travelers, who share our resentment of devoting the majority of holiday time for taking pictures and recording videos. The quadcopter cameraman is a First-Person-View (FPV) quadcopter that is designed for easily capturing the environment around the proximity of the traveler. It is the first step towards the ultimate goal of fully automation of the work done by a professional cameraman, so that an average person can have his or her time of life well documented from a brand new angel previously unavailable without the help of quadcopters.

To achieve this goal, the quadcopter in this project incorporates a series of design that enable a user-friendly interface, robust camera functions, and most importantly, an agile quadcopter system that is able to perform flexible maneuvers and could be operated safely in indoor and outdoor environments. The quadcopter cameraman is composed of a FPV and a PC communication base the quadcopter connected to via WIFI, which sends commands to and receives data from the quadcopter. The quadcopter is able to be fully controlled from the platform. It can take off vertically and hover at a stable position before being commanded to land. The quadcopter is expected to fly between 2 to up to 20 minutes in the air depending on the complexity and intensity of maneuvers involved. Upon receiving command from the user, the quadcopter could perform picture-taking and other photographic functions. Additionally, the quadcopter supports a precise discrete 360 degrees turn maneuver, similar to a stepper motor in order to create panorama images. While in the air, the quadcopter is able to capture videos and store them in the on board PC. The quadcopter itself is designed to be as light as possible in order to maintain a long hovering time. Additionally, the quadcopter system itself is flexible to additional features as the whole system develops and updates.

### Specification of External Environment

The quadcopter need to operate in an industrial environment in a commercial grade temperature. Also, the quadcopter need to operate in an indoor environment which provides stable and high-bandwidth WIFI signal coverage to be able to communicate with the drone. The weather condition for this project is assumed to be relatively mild, which means, no strong wind, rain, snow. The temperature is assumed to be varying around room temperature and so does humidity.

Commercial grade: 0 °C to 70 °C

## System Input and Output Specification

### System Inputs

*User inputs*

Quadcopter Cameraman system supports both manual and autopilot system for user navigation input. Even though having the word "manual" in manual control's name, both manual and autopiloting are densely computer aided. The reason behind this is that a quadcopter is intrinsically aerodynamically unstable and without proper control it would easily crash. As a result, the user inputs are restricted by a number of states so that an chances of an undefined user input(For example, take-off command when the quadcopter is in the air) are minimized. Besides Flight control commands, the user can freely perform any photographic functions by giving photographic commands including taking pictures, record videos and initiate live streaming.

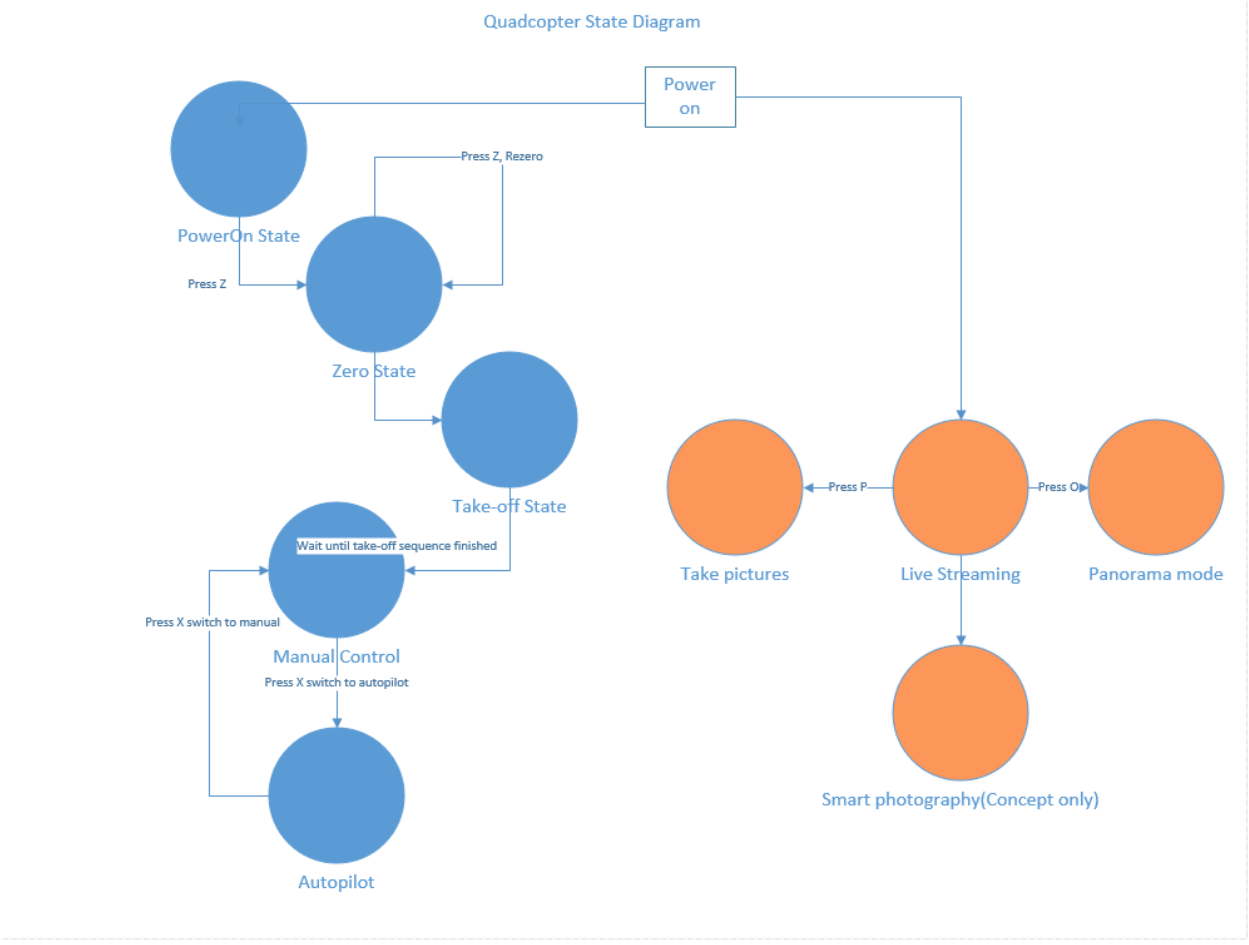



**Figure 3.1 Flow Chart of User Input**

All user controls are accepted from a standard keyboard. Besides a number of keys available throughout the operation. Most keys are only available, or mean different commands at different states of the quadcopter.

Global key presses that are available all the time include the ‘ESC’, which exits the flight control program, crash landing(‘c’), which reset the quadcopter, picture-taking(‘p’), which takes a single picture’), record(‘v’), which start recording a video, and panorama(‘u’, not implemented).

Before takeoff, the user is presented with the ground operation interface. The user is free to zero (auto calibrate) the quadcopter, or re-zero the quadcopter by pressing ‘z’ on a standard keyboard. The user can press ‘m’ to enter or exit motor test mode. The quadcopter can and only can enter motor test mode in ground operation, as such operation is dangerous when it is hovering on the air. A keyboard press of ‘T’ exits the ground control interface and initiate the take-off sequence, which is a sequence of actions precisely timed to efficiently, and safely take off the quadcopter.

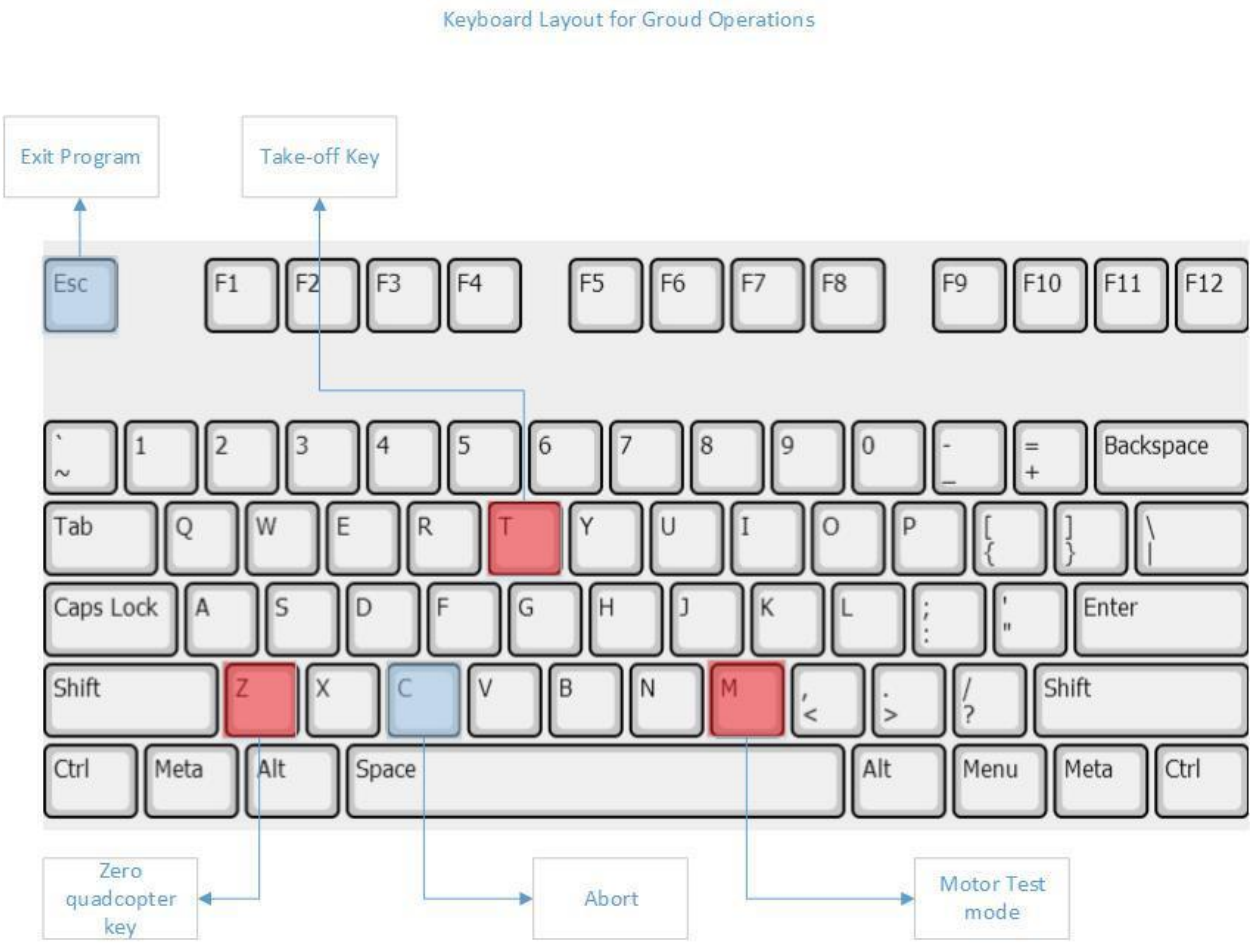



**Figure 3.2 Keyboard Layout for Ground Operations**

Similarly, in the motor test mode, user and increase the speed of motor1, 2, 3, 4, using QWER keys respectively and they can decrease the speed of 4 motors using ASDF keys.

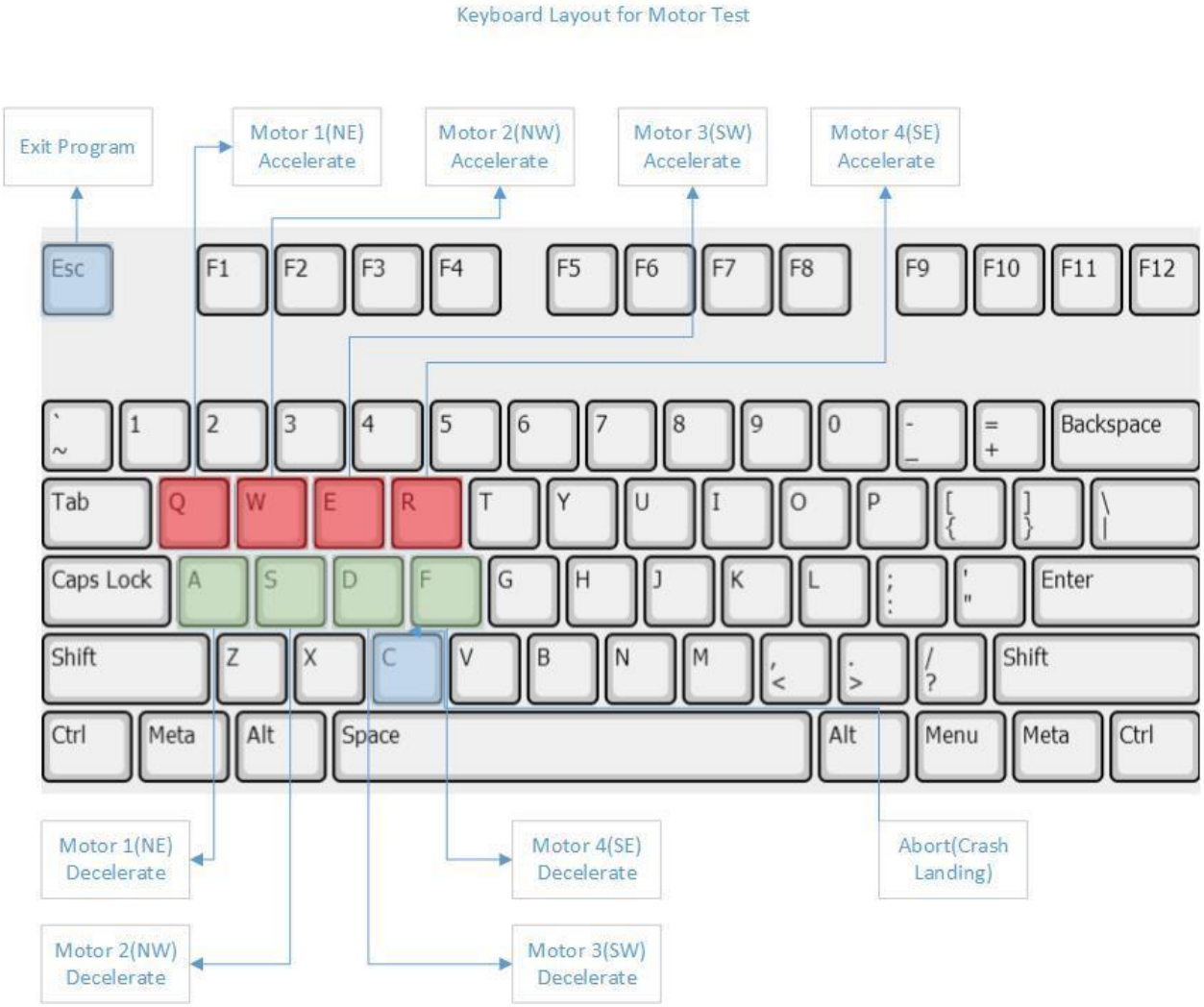



**Figure 3.3 Keyboard Layout for Motor Test**

In the manual control state, the quad copter could be controlled using the WASD keys. The control scheme is similar a common first person shooter video game so that users can quickly adapt to the control. W and A keys control the roll, A and D control the pitch. Q and E control the yaw. RF control the throttle. X switches between autopilot and manual control. Space auto balance the quadcopter (clear user control offsets so that the quadcopter is in upright position). L initiate a landing sequence that safely and efficiently land the aircraft (not implemented).

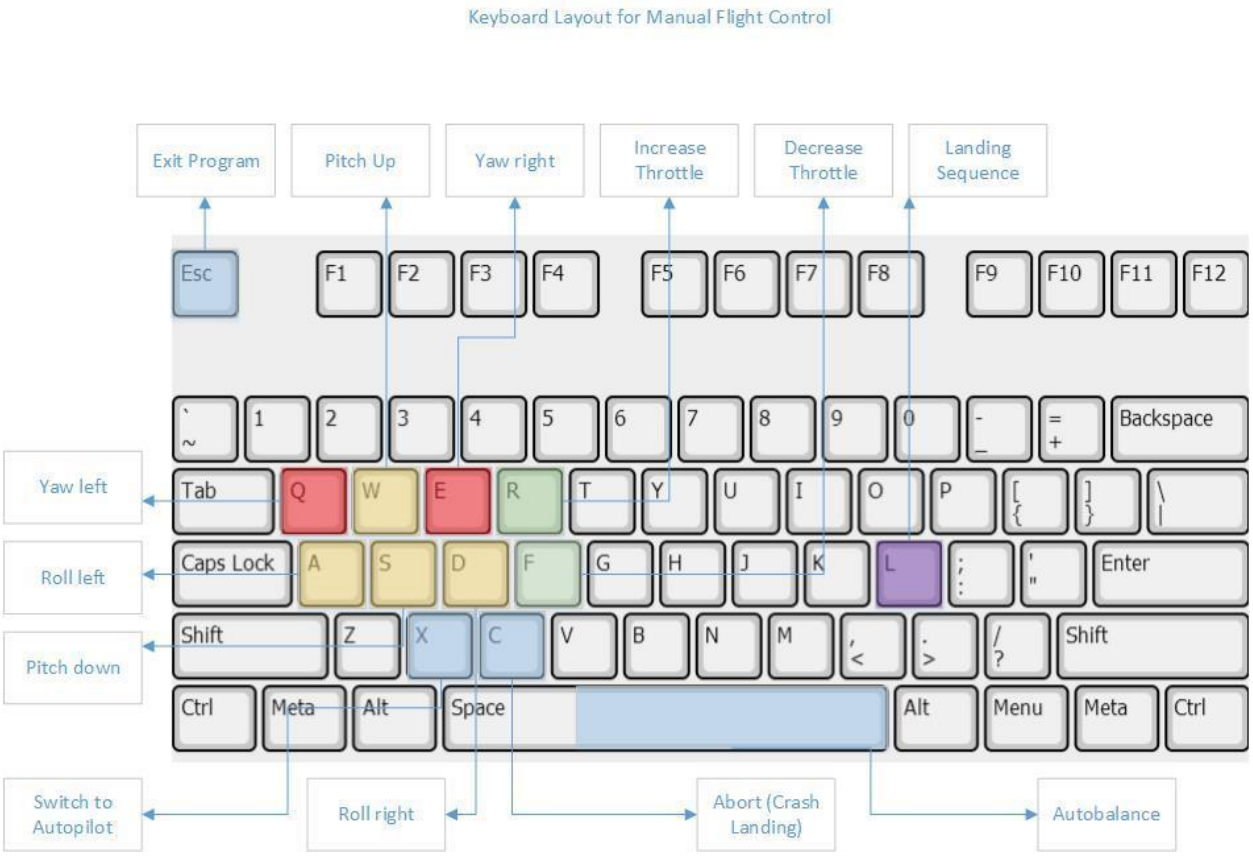



**Figure 3.4 Keyboard Layout for Manual Flight Control**

The autopilot control interface is similar to the manual one asides from the fact that WASD+ERRF would be interpreted as going forward, backward, left, right, rotate left, rotate right respectively instead of changing Euler angles of the quadcopter directly. Due to time constraints of the project, the autopilot option was not explored and implemented.

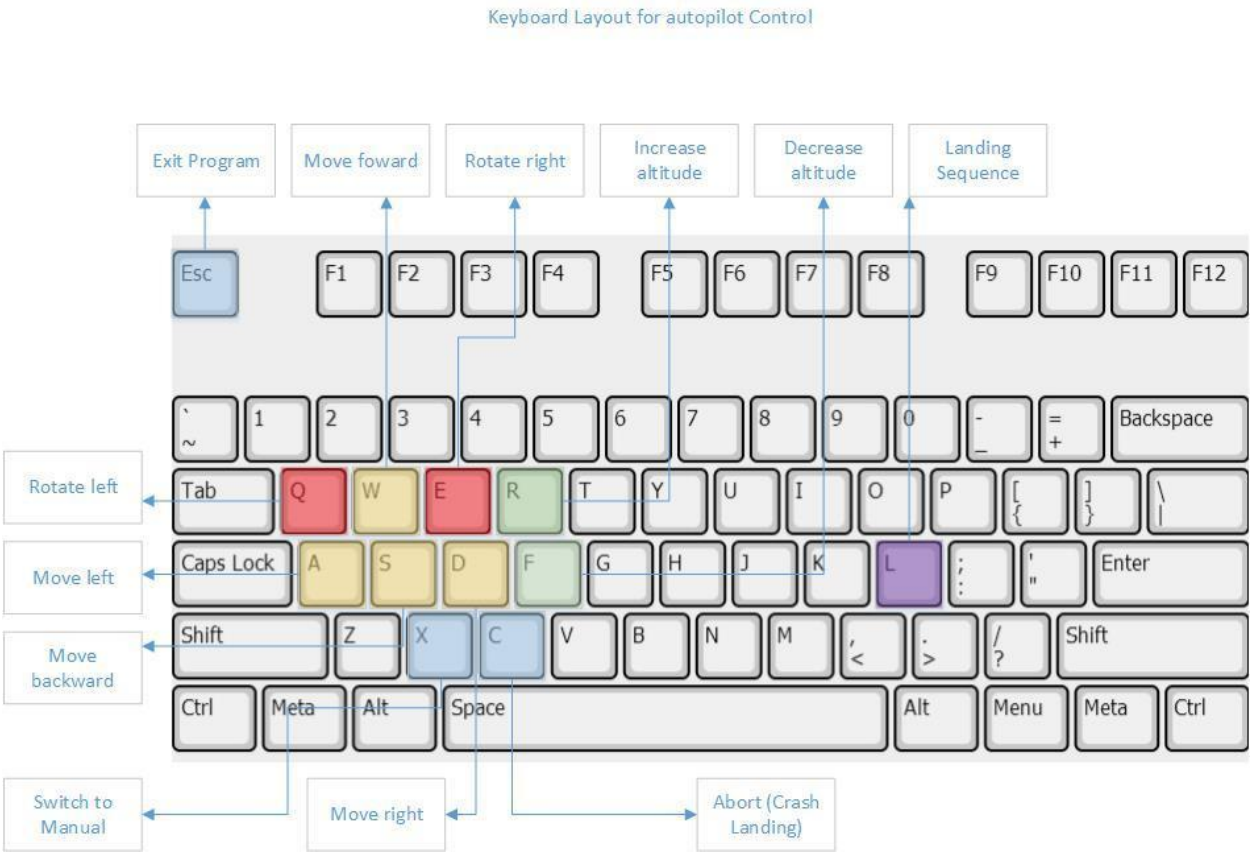



**Figure 3.5 Keyboard Layout for Autopilot Control**

**Environmental Inputs**

Asides from the user input, the system takes a number of environmental inputs mainly for feedback control. The environment inputs are specified below.

Range to frontal objects ranging: up to 3 m ± 0.1m (Not implemented).
Barometric pressure changing from: altitude 33.5m(Seattle ground) to 100m, ±15cm (Not implemented).
Angular position of quadcopter in its pitch axis: 30° ± 1.0°
Angular position of quadcopter in its roll axis: 30° ± 1.0°
Angular position of quadcopter in its yaw axis: 0°-360° ± 1.0°

Acceleration: for all axis 10g ± 0.1g
Environmental Temperature: 0 to 100C°, ±1C° (Not implemented).

**System Outputs**

Physical locomotion of the quadcopter corresponding to the user input via the flight control interface at 10Hz
Updating a website live streaming pictures and videos at 100kbytes

**User Interface**

The user will be able to type in different command through Secure Shell (SSH) from remote computer. The quadcopter will then be able to respond to each of the command.

**Mode**:

*Hover, Capture, Record, Land, Zero*

**Reset**:

The reset button will reset the electronic component on the board.

The following screenshot shows the interface for clients to control the quadcopter. The example shows the Input "Z" which represents "Zeroing" the quadcopter. If the sent command does not exist, it will output the line "Invalid Command" as shown below.

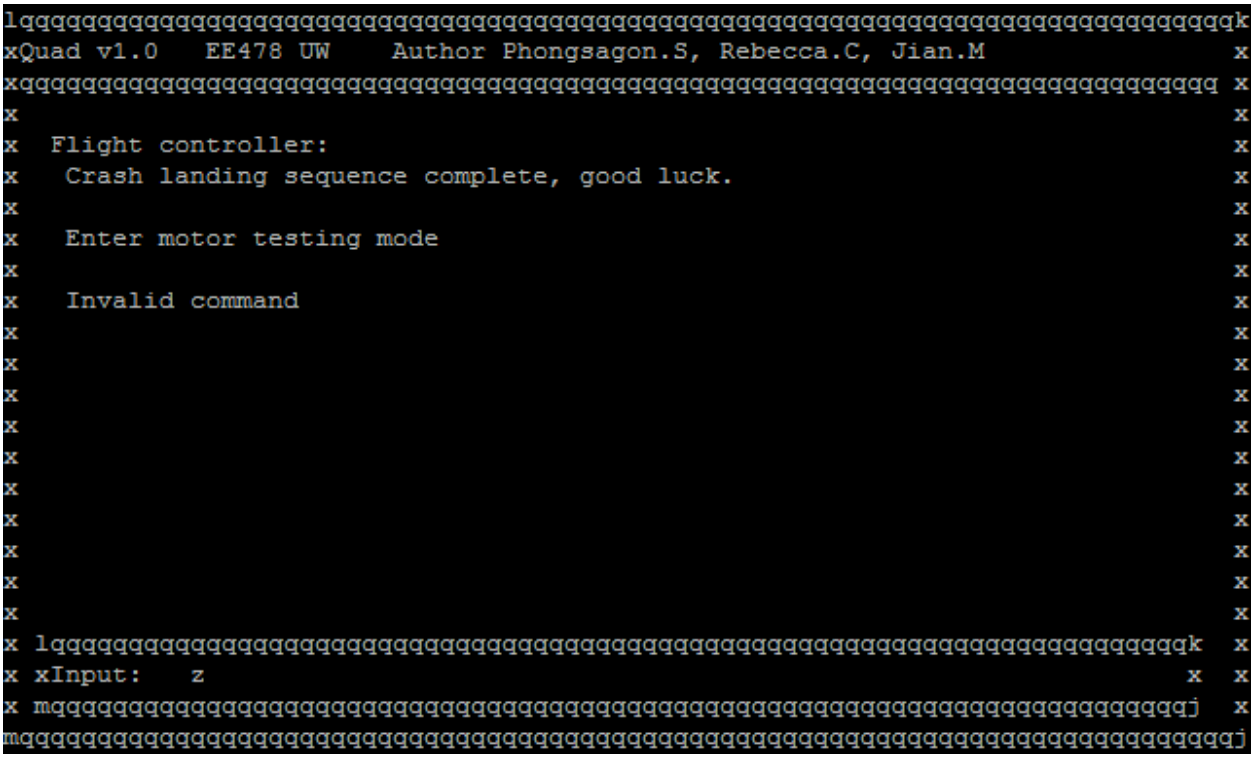



**Figure 3.6 Screen For User Control**

## Use Case

The use case below shows the features quadcopter can do as requested from the user from the PC. The user is represented by the human figure in the use case below. Each bubble connected to the user is the commands the quadcopter are capable of.

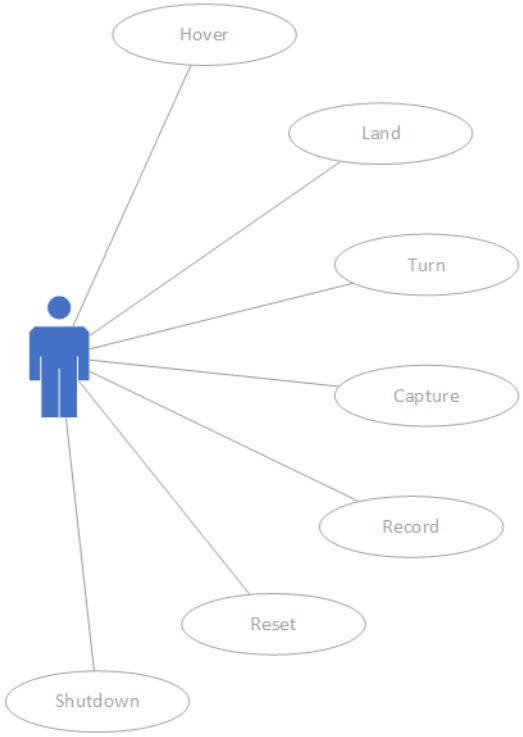



**Figure 3.7 Use Case Diagram for Quadcopter System**

***Hover***

The user sends hover command and specifies desired altitude from the PC and the quadcopter is going to fly off the starting platform and elevate to the requested height and remain a stable position. The quadcopter is going to suspend in air for a maximum of 20 minutes, the maximum amount of time the battery can support the hoving feature.

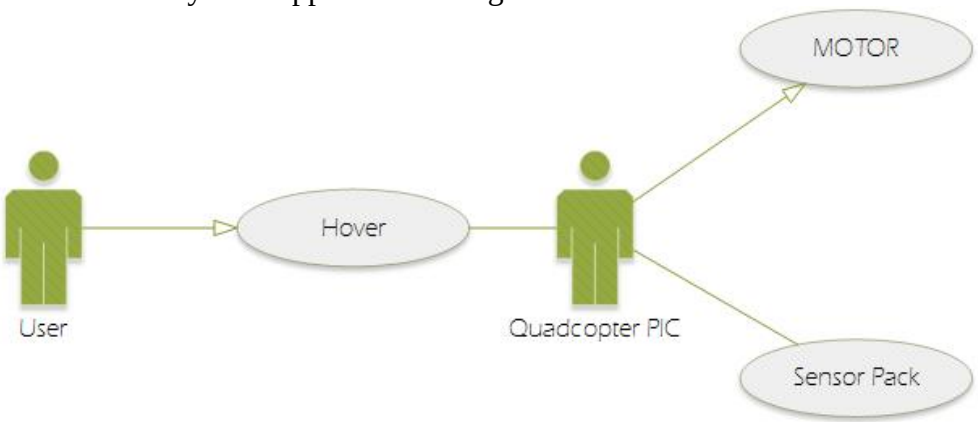



**Figure 3.8.1 Use Case Diagram for Hover**

***Land***

The "Land" command tells the quadcopter to descend from the original flight position back to the original departure platform. The landing feature will allow the quadcopter tp safely drop in its' altitude back to the ground where the four motor driven propellers will cease all it's rotations. The quadcopter will terminate all it's features exercised in the Hover mode.

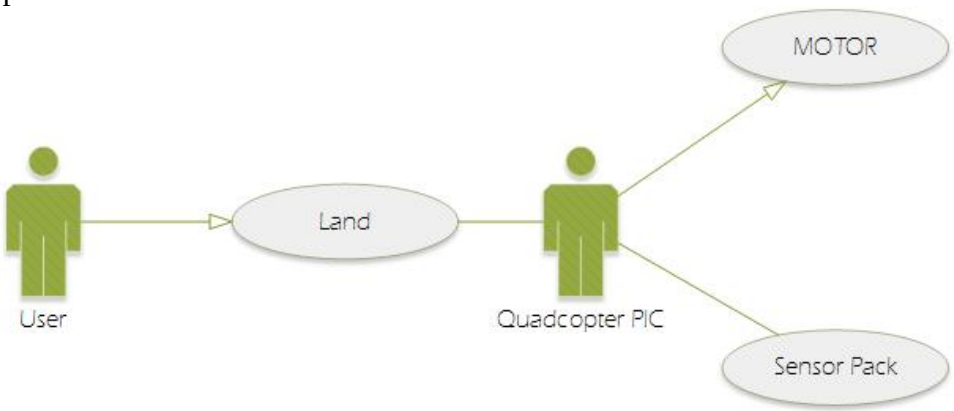



**Figure 3.8.2 Use Case Diagram for Land**

***Turn***

The "turn" command is only valid when the quadcopter is in the hovering state. This command tells the quadcopter to make a 360° turn mid-air. The turn can support either turn in counterclockwise rotation or clockwise rotation, determined upon the user's request.

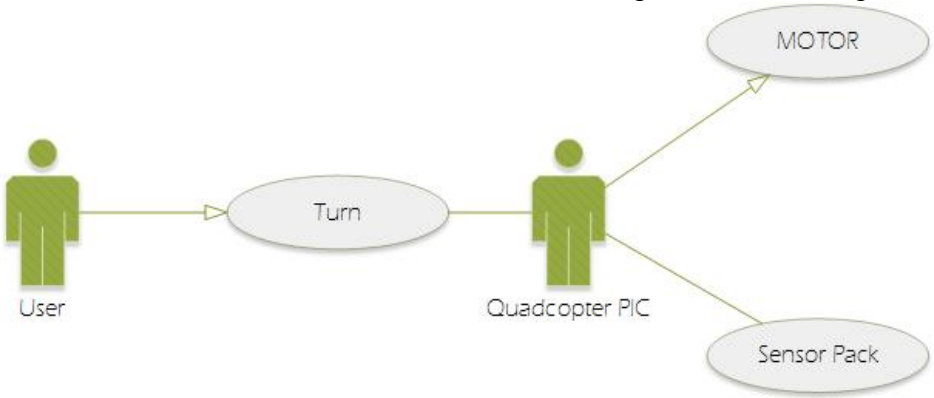



**Figure 3.8.3 Use Case Diagram for Turn**

***Capture***

If the user sends a "Capture" command, the quadcopter will activate the camera on the quadcopter to taking a picture at the exact location the quadcopter is at. The camera used is the Raspberry Pi Camera and the captured camera is going to be sent back to the user to view on the PC. The capture feature can be activated either when the quadcopter is in flight or is stationary on ground.

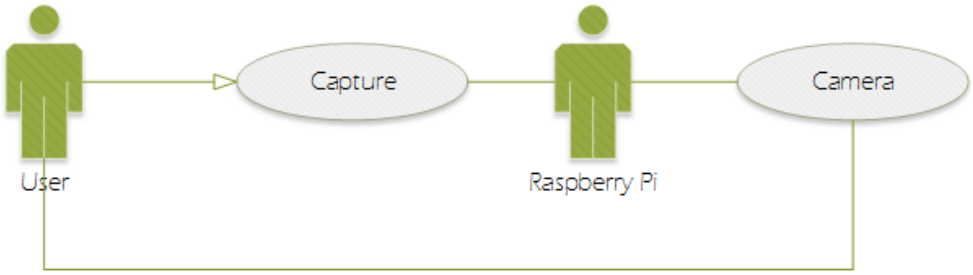



**Figure 3.8.4 Use Case Diagram for Capture**

***Record***

If the user sends a "Record" command, the quadcopter will activate the camera on the quadcopter to to start recording the video at the exact location the quadcopter is at. The camera used is the Raspberry Pi Camera and the captured camera is going to be sent back to the user to view on the PC. To terminate the video recording, the user needs to send another record stop command. The capture feature can be activated either when the quadcopter is in flight or is stationary on ground.

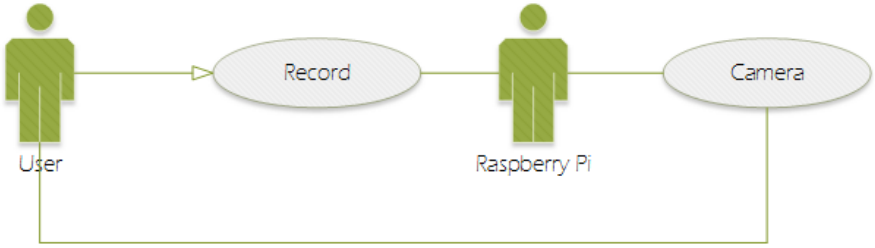



**Figure 3.8.1 Use Case Diagram for Record**

***Reset***

The Reset feature is to calibrate all the sensors on the quadcopter such as barometer and gyroscope. These sensors determines the altitude and positioning of the quadcopter. The sensors will be leveled in respect to the quadcopter ground position. For example, the quadcopter determines its' height through finding the difference in distance between the start and the current position. Therefore, reset feature is to zero the starting distance. As a result, the sensor on quadcopter will have a more accurate measurement of relative location from the starting off platform.


**Figure 3.8.1 Use Case Diagram for Reset**

## System Functional Specification

## Flight Control System Architecture

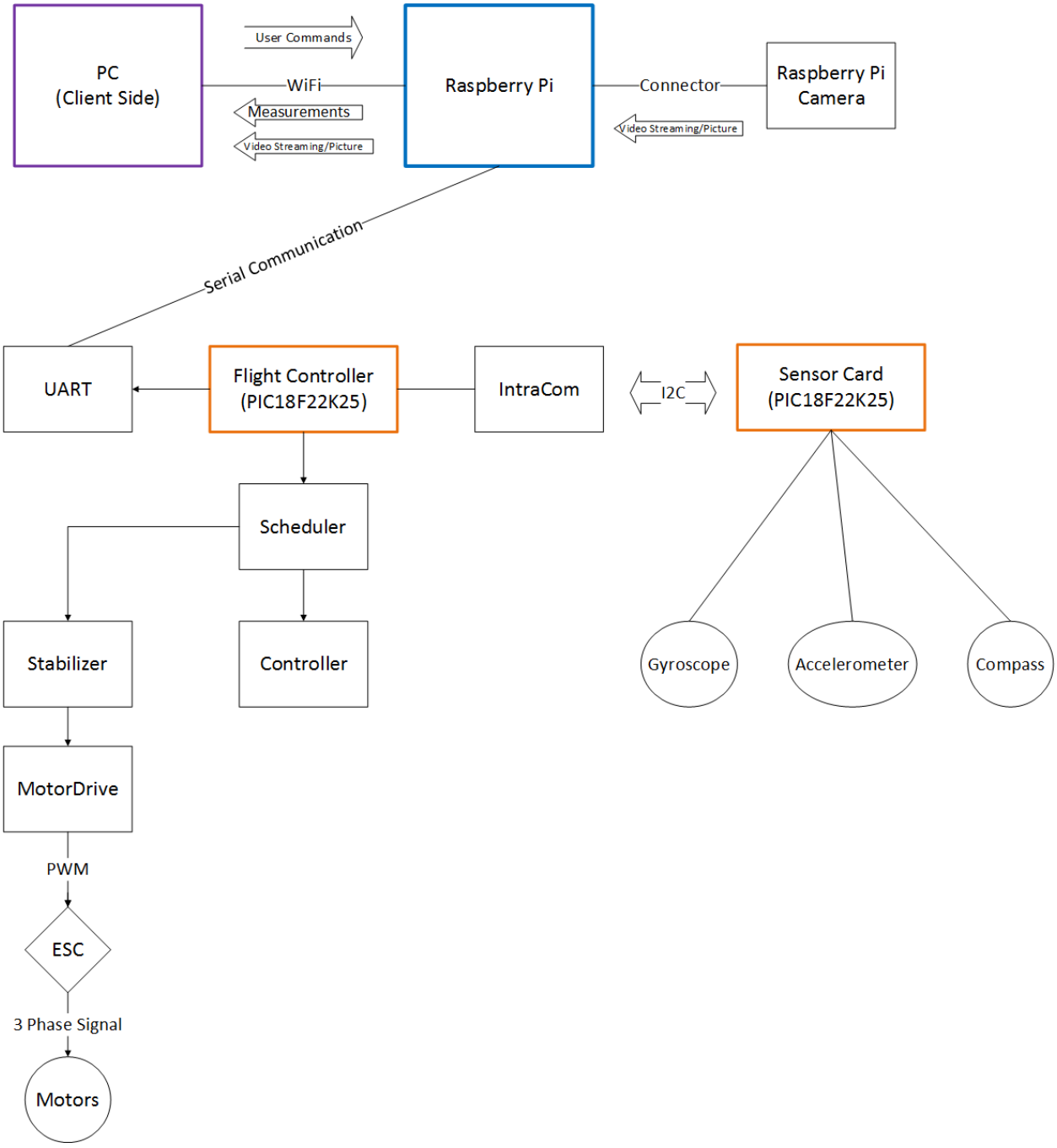



**Figure 3.9 Overall System Diagram**

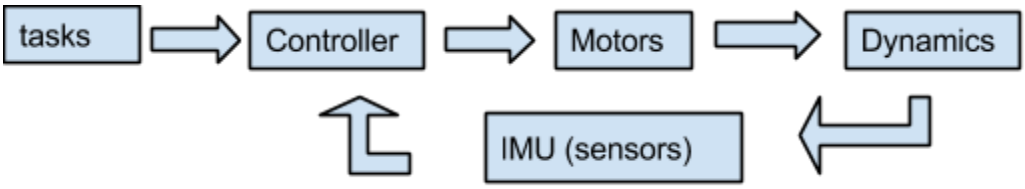



**Figure 3.10 Task Block**

The task block represents the input command from the users. The tasks can be changed by the command of the user such as hover and land through secure shell from the remote computer to Raspberry Pi. The Raspberry Pi then decode the command then send the signal to the Master pic through UART.

The control block represents the control task to stabilize the quadcopter. From the sensors, it is possible to have a feedback on the position and provide the automatic stabilization.

The motors block compose of the power board and ESCs which provides the voltage to various components in the system. The Electronic Speed Controller controls the speed of the motor according to the controller.

The dynamic block refers to the propeller's velocity in reference to the initial position of the quadcopter.

The IMU block provides the information about the quadcopter' attitude and heading. It has three-axis accelerometer, three-axis gyroscope, three-axis compass and a barometer. The IMU would provide enough information to calculate roll-pitch-yaw angles and send them through I2C from the slave Pic to master Pic.

**Power**

The power supply will provide the voltages at the specified current levels to various component. On the quadcopter, the power board will be mainly supplying power to: 1 Sensor pack, 4 motors, 1 Raspberry Pi, 2 PICs, and 4 ESCs.

| **Component** | **Voltage** | **Current** |
|---|---|---|
| Sensor Pack | 5.0V/3.3V | |
| Motor | 5.0V | |
| Raspberry Pi | 5.0V | |
| PIC | 5.0V | |
| ESC | 11.1V | 10A |

**Table 3.11 Power Requirements for each component of Quadcopter**

The entire quadcopter and its' components are powered by the 7A Lithium Polymer Battery which outputs 11.1V. The Electronic Speed Controllers (ESC) that controls the four motors will be powered with 11V, straight from the battery pack. The remaining electronics on the quadcopter are all supplied with 5V. The 5V supply is achieved with a 5V voltage regulator that inputs 11.1V from the battery pack and outputs 5V. The 5V voltage regulator is a step down buck switching regulator.

**High Level Diagram**

Upon power on the quadcopter, the system will stay in IDLE mode for command. The command will be received in Raspberry Pi through Secure Shell (SSH) protocol from a remote computer. The system will have a closed loop feedback control for the stability and flight control of the quadcopter from the input of the collected data from sensors. If the capture command is entered, the Pi camera will respond to the command accordingly.

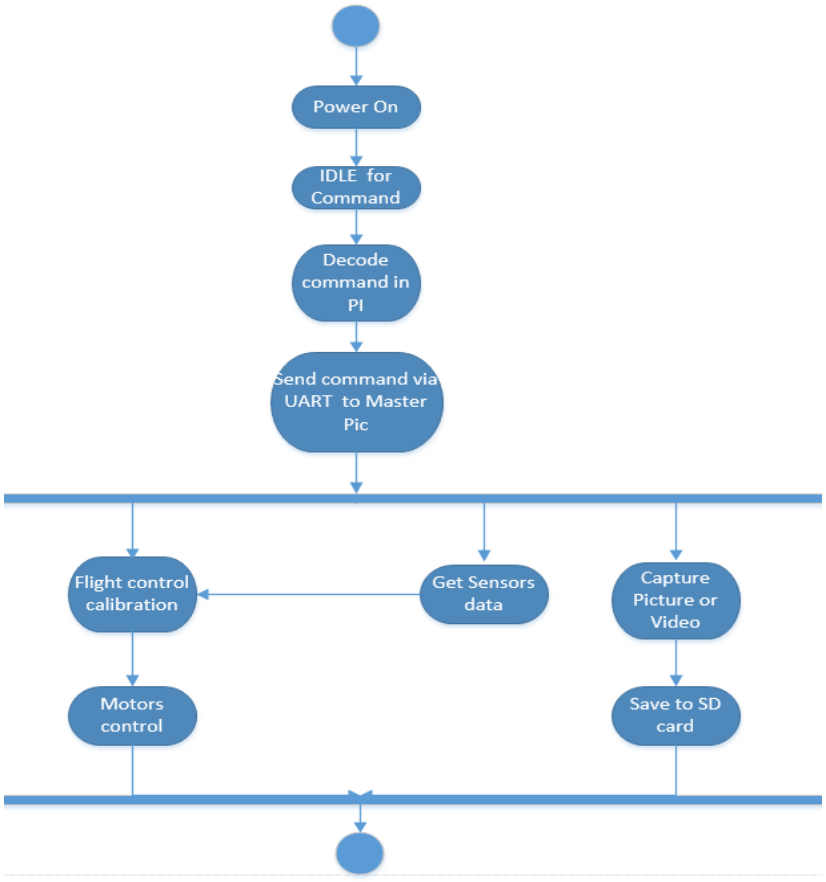



**Figure 3.12 Functional decomposition for Quadcopter System**

## Operating Specification

Temperature Range: 20-50 C° typical, 90 max.
Humidity up to 90% RH non-condensing
Battery charging 120 – 240 VAC 50 Hz, 60 Hz, 400 Hz, 15 VDC
The system shall operate for a minimum of 15 mins on a fully charged battery
The system time base shall meet the following specifications

## Reliability and Safety Specification

The quadcopter should have full blade protection for propellers and an emergency landing plan.
The quadcopter should be compliant with the AMA National Model Aircraft Safety Code.
The use of quadcopter should be compliant to state with federal privacy laws regulations.

## DESIGN PROCEDURE

We designed a flight controller, power regulation module, a sensor module for the quadcopter.

## Flight Controller

Since the quadcopter is aerodynamically unstable, a robust controller determines if the aircraft crashes or not. The main design concern for designing a quadcopter controller is to stabilize the quadcopter at any time while giving users, or the autopilot software, the freedom to control it. The initial stage of such task starts with creating a simplified model of the real world physics involving the quadcopter. We researched several mathematical model describing mechanical and aerodynamic property of the quadcopter, and decided the apply

After figuring out the mechanical and aerodynamic property of the quadcopter, we incorporate the physics model into a feedback loop that turns the system into a stable system.

Manual control of roll, pitch and yaw is done by adjusting the expected output value of the PID controllers. The throttle

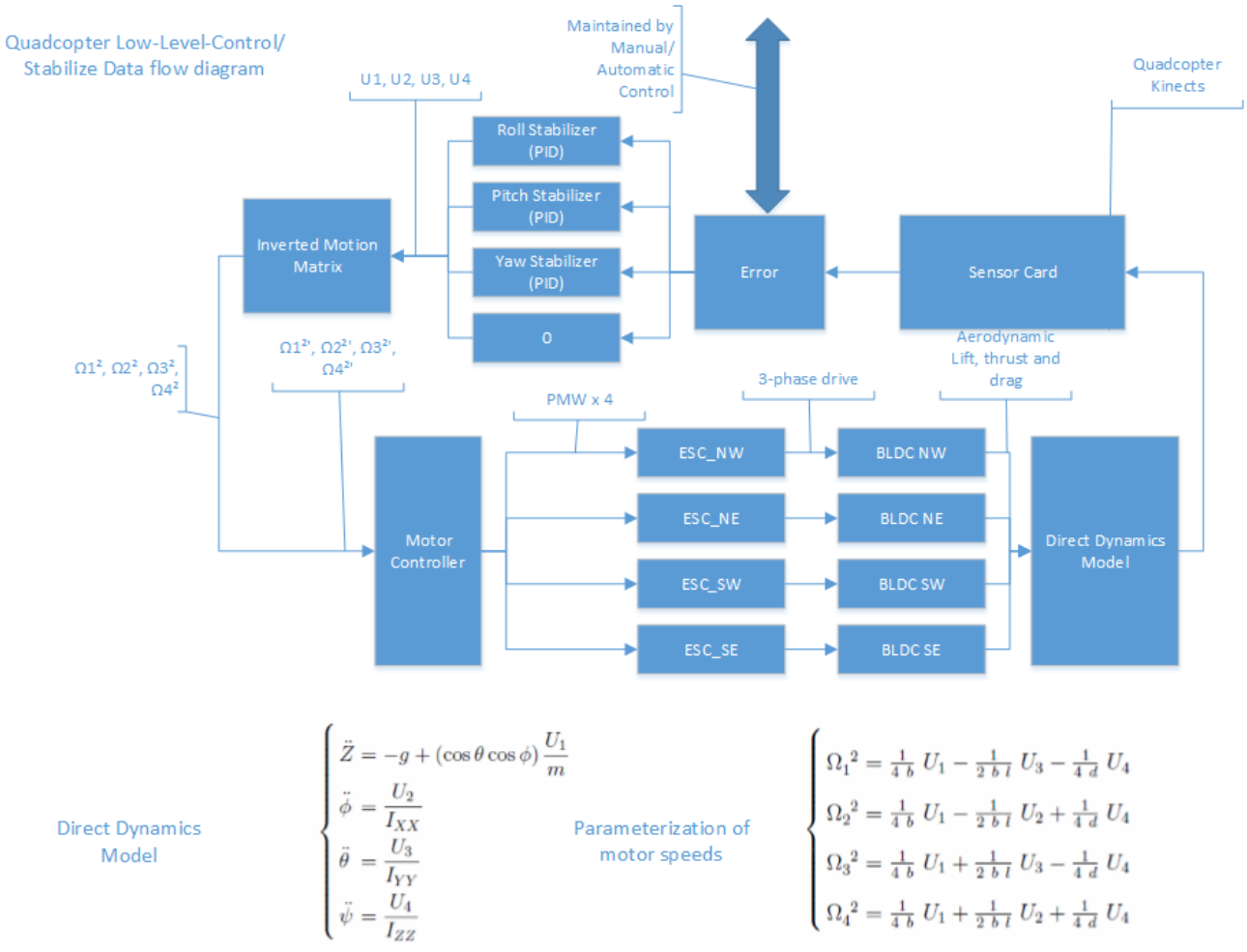



**Figure 3.13 Overall Flight Control for Quadcopter**

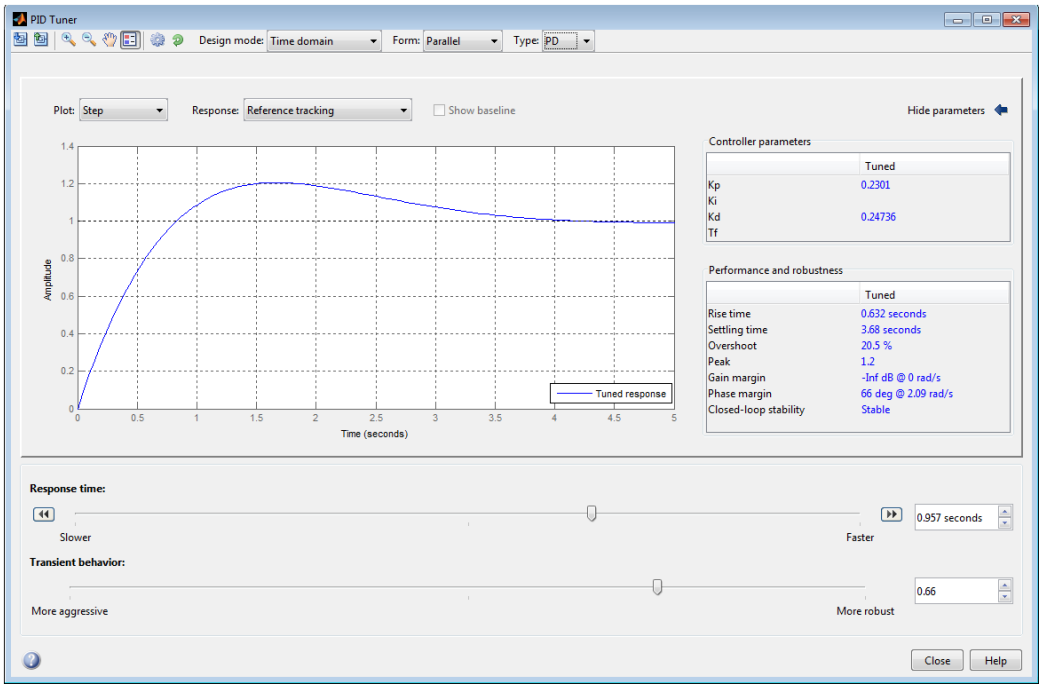



**Figure 3.14 Pitch and Roll PID Tuner from MatLab**

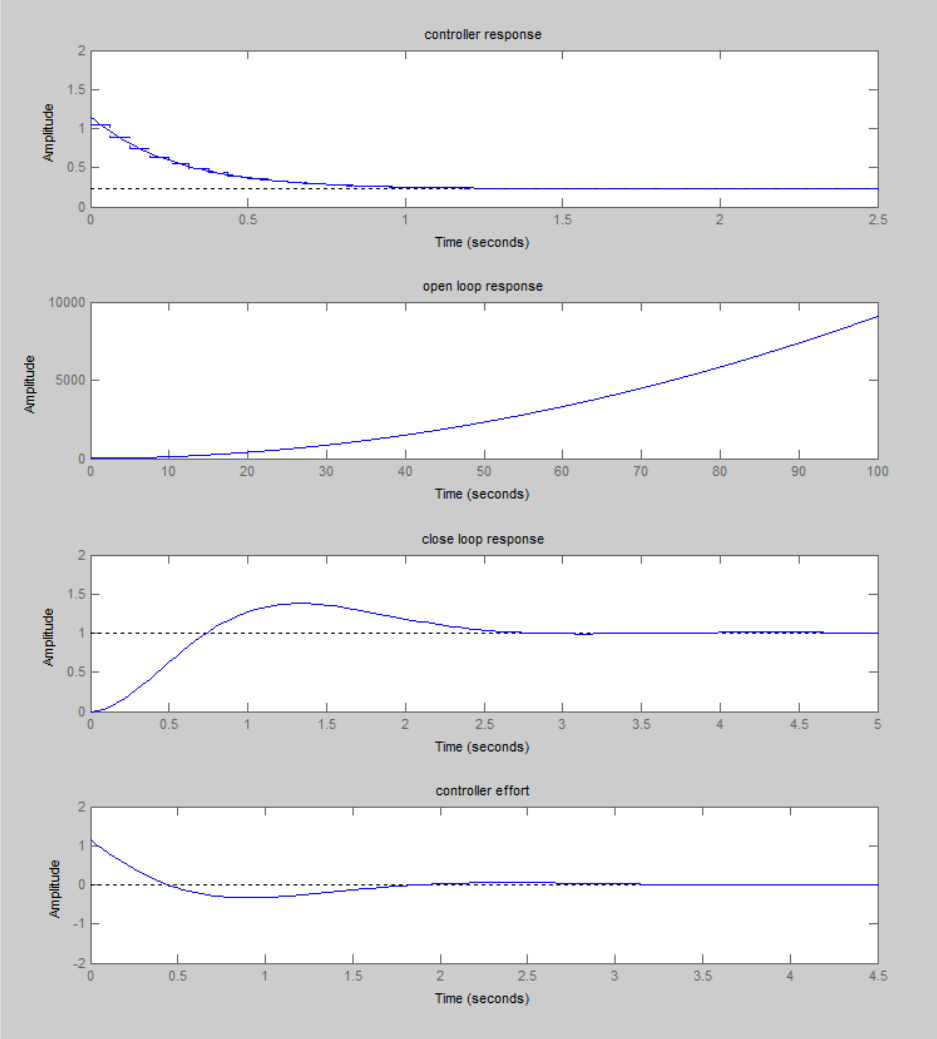



**Figure 3.15 Pitch and Roll Step Response**

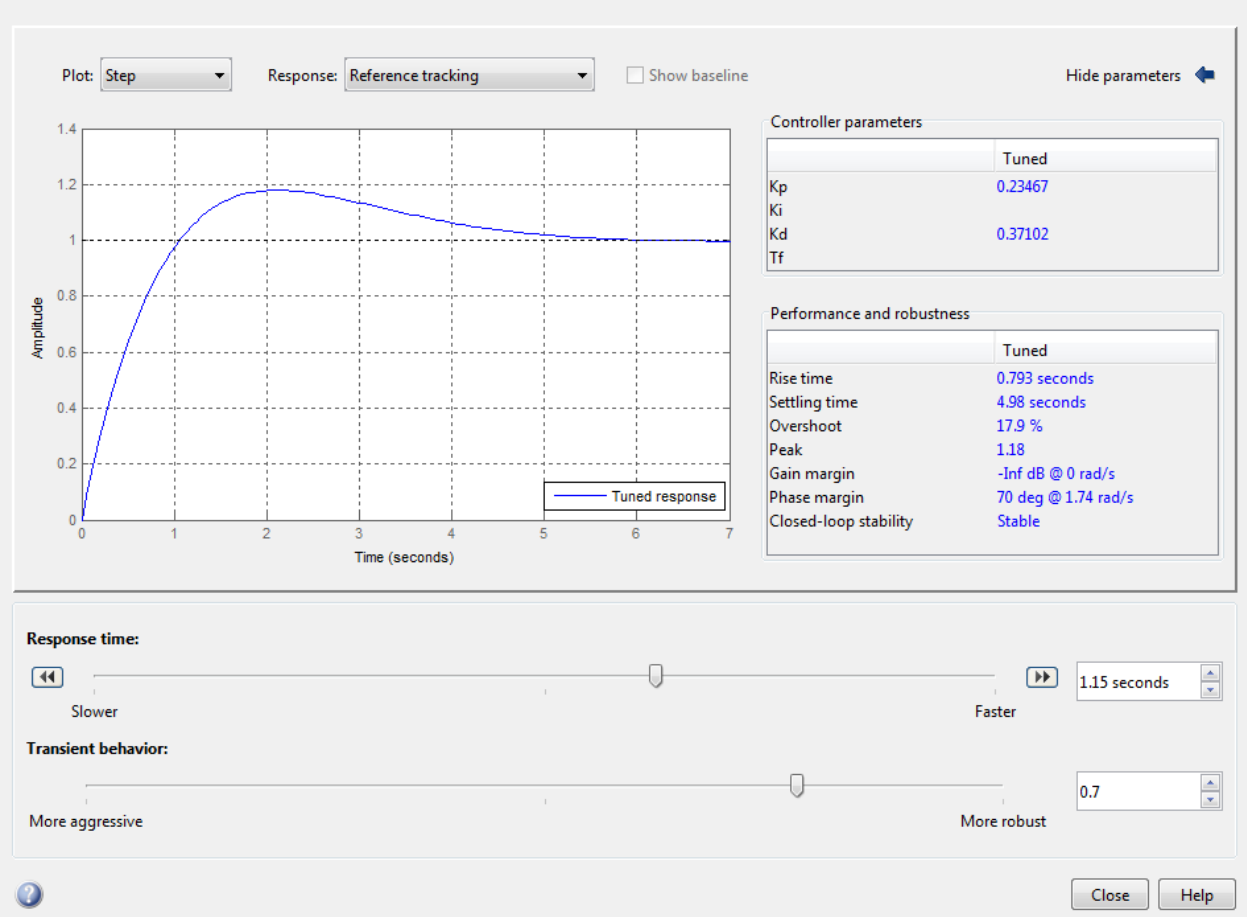



**Figure 3.16 Yaw PID Tuner from MATLAB**

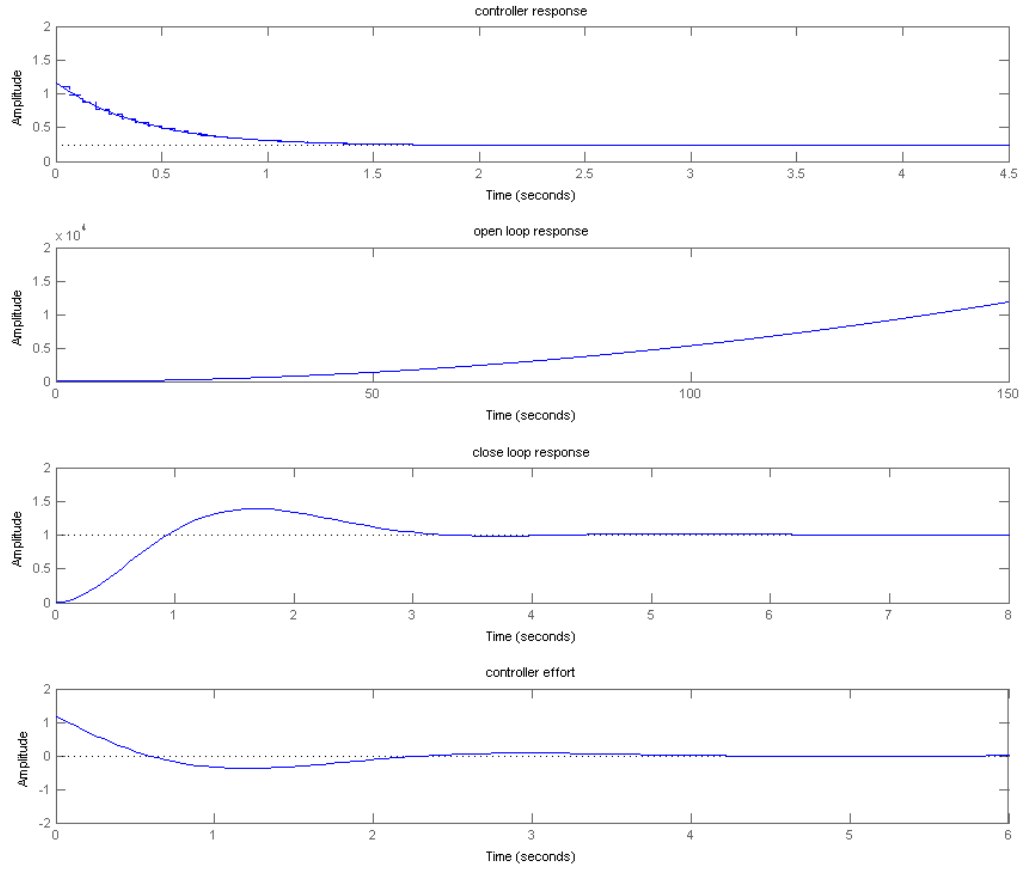



**Figure 3.17 Yaw Step Response**

## Sensor Card

Our sensors card consist of accelerometer and gyroscope. We could use both of these to measure the pitch and roll. However, there is a limitation to these sensors.

The problem with accelerometer. Every small forces will disturb the measurement of the data so it will fluctuated really fast. The data is reliable only on the long term.

The problem with gyroscope. In order to get the angles, we need to integrate the angular positions over time. However, it has the tendency to drifts, meaning the data is not returning to zero when it is at original position. Thus it is reliable only on short term.

The Kalman filter is too complicated, considering the period of time we have. Thus we decided to use the complimentary filter to integrate the sensors.

The main idea about complementary filter is to pass the digital low pass filter to accelerometer while pass the high-pass filter to gyroscope data. The equation can be seen below.

$$angle = 0.98 * (angle + gyrData * dt) + 0.02 * (accData)$$

dt- the time step for the integration of gyroscope

This filter is light and can be easily implemented compared to the Kalman filter and Direct Cosine Matrix, making it ideal for the embedded system, with PIC18.

## SYSTEM DESCRIPTION

The quadcopter system is composed of a flying quadcopter connected to its PC communication base, which send commands to and receives data from the quadcopter. The flying quadcopter is able to fly above the platform and hover at a stable position before being commanded to land. The quadcopter is expected to fly between 2 to up to 20 minutes in the air. Upon receiving command from the user, the quadcopter could perform picture-taking and other photographic functions. Additionally, the quadcopter should support a precise 360 turn maneuver in order to create panorama images. While in the air, the quadcopter is able to capture videos and store them in the Pi. The quadcopter itself should be lightweight in order to maintain a long hovering time. Additionally, the quadcopter system itself should be flexible to additional features as the whole system develops and updates.

### Specification of External Environment

The quadcopter operates in stable indoor environments, i.e, obstacles mainly constitute walls, passengers and various daily objects. The working environment needs to provide stable and high-bandwidth WIFI signal coverage to the quadcopter the Internet. If the environment displays a characteristics of low signal strength or low bandwidth, the system would only be partially functional, and execute the emergency landing plan. The weather condition for this project is assumed to be relatively mild, which means, no strong wind, rain, snow. The temperature is assumed to be varying around room temperature and so does humidity.

Commercial grade: 0 °C to 70 °C

### System Input and Output Specification

### Overall System Inputs

The system the following environment inputs

Range to frontal objects ranging: up to 3 m ± 0.1m.
Barometric pressure changing from: altitude 33.5m(seattle ground) to 9000m, ±15cm.

Angular orientation of the quadcopter: for all axis up to 30° ± 0.01°
Angular velocity: 0 to ±1/80π
Acceleration: for all axis 10g ± 0.1g
Environmental Temperature: 0 to 100C°, ±1C°.

The system takes the following user inputs

Flight commands:

Zero quadcopter
Change altitude [expected altitude]
Start recording
Stop recording
Take single pictures
Take a panorama view (automatic rotation)

The user will be able to type in different command through Secure Shell (SSH) from remote computer. The quadcopter will then be able to respond to each of the command.

Mode:

***Hover***, ***Capture***, ***Record, Reset, Shutdown, Land, Turn***

**Overall System Outputs**

Physical roll, pitch and yaw change
Updating a website live streaming pictures and videos

**Bill of Materials**

| Part role | Part function | Vendor | (Vendor) PN | Unit Cost | # needed per unit | Total Cost |
|---|---|---|---|---|---|---|
| Microcontroller | Flight controller, sensor controller | Microchip | PIC18F25K22 | $2.87 | 2 | $5.74 |
| On-board PC | wireless comm and video processing | N/A | Raspberry PI model A | $30.00 | 1 | $30.00 |
| Motors | BLDC motors x 4 | SunnySky | A2212-800 | $14.56 | 4 | $58.25 |
| Battery | Power source | Turnigy | Turnigy 5000mAh 3S 25C Lipo Pack | $28.80 | 1 | $28.80 |
| Camera | Multimedia functions | N/A | N/A | $34.49 | 1 | $34.49 |
| Misc Machine screws /nuts/washers | Connect the main frame | Hardware store | N/A | $0.07 | 20 | $1.40 |
| Header Pins | Misc. Connectors | Black market | N/A | $0.01 | 20 | $0.20 |
| Long screws & washers | Main board | Hardware store | N/A | $0.05 | 4 | $0.20 |
| Electric Speed Controller | motor controller | Hobby Wing | 10A | $11.99 | 4 | $47.96 |
| Aluminium net | Edge protection | Hardware store | N/A | $0.25 | 4 | $1.00 |
| Wifi adapter | N/A | RT8188C0 | N/A | | | $10.95 |
| Propeller CW, CCW(pair) | N/A | N/A | APC propeller | $11.04 | 1 | $11.04 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Magnetic Sensor(HMC 5883L breakout board) | Honeywell | Parallax Inc. | 29133PAR-ND | $30.00 | 1 | $30.00 |
| GY521(MPU 6560 breakout board) | Invensense | Invensense | GY521 | $10.00 | 1 | $10.00 |
| LEDs | System status indicator | N/A | N/A | $0.50 | 4 | $2.00 |
| 4.7K Resistors | Various | N/A | N/A | $0.10 | 15 | $1.50 |
| Capacitors | Various | N/A | N/A | $0.10 | 15 | $1.50 |
| Perf Board | PCB | SHUNDA | 7cm x 9cm | $6.98 | 1 | $6.98 |
| Total | | | | | | $282.01 |

**Table 3.18 Bill of Materials**

**Use Cases**

The use case below shows the features quadcopter can do as requested from the user from the PC. The user is represented by the human figure in the use case below. Each bubble connected to the user is the commands the quadcopter are capable of.

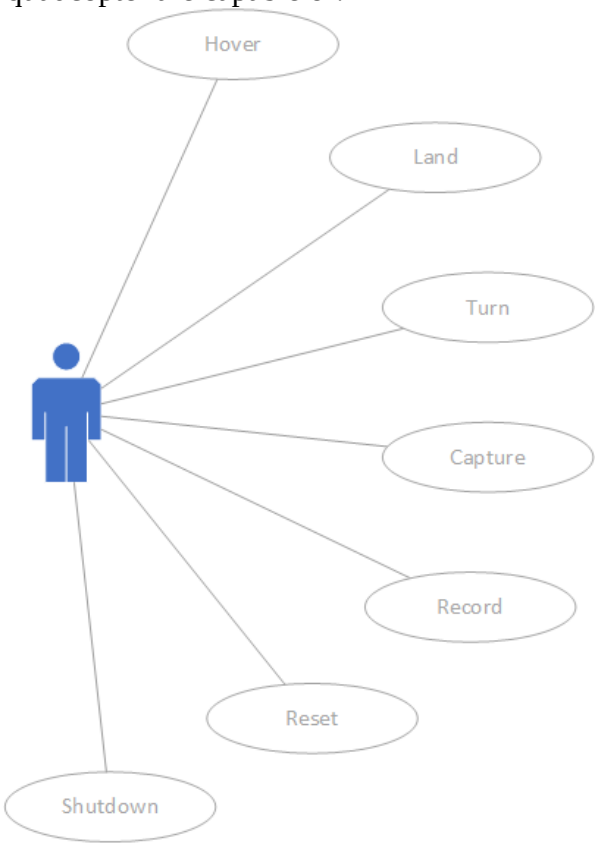



**Figure 3.19 Use Case Diagram for Quadcopter System**

***Hover***

The user sends hover command and specifies desired altitude from the PC and the quadcopter is going to fly off the starting platform and elevate to the requested height and remain a stable position. The quadcopter is going to suspend in air for a maximum of 20 minutes, the maximum amount of time the battery can support the hoving feature.

***Land***

The "Land" command tells the quadcopter to descend from the original flight position back to the original departure platform. The landing feature will allow the quadcopter tp safely drop in its' altitude back to the ground where the four motor driven propellers will cease all it's rotations. The quadcopter will terminate all it's features exercised in the Hover mode.

***Turn***

The "turn" command is only valid when the quadcopter is in the hovering state. This command tells the quadcopter to make a 360° turn mid-air. The turn can support either turn in counterclockwise rotation or clockwise rotation, determined upon the user's request.

***Capture***
If the user sends a “Capture” command, the quadcopter will activate the camera on the quadcopter to taking a picture at the exact location the quadcopter is at. The camera used is the Raspberry Pi Camera and the captured camera is going to be sent back to the user to view on the PC. The capture feature can be activated either when the quadcopter is in flight or is stationary on ground.

***Record***
If the user sends a “Record” command, the quadcopter will activate the camera on the quadcopter to to start recording the video at the exact location the quadcopter is at. The camera used is the Raspberry Pi Camera and the captured camera is going to be sent back to the user to view on the PC. To terminate the video recording, the user needs to send another record stop command. The capture feature can be activated either when the quadcopter is in flight or is stationary on ground.

***Reset***
The Reset feature is to calibrate all the sensors on the quadcopter such as barometer and gyroscope. These sensors determines the altitude and positioning of the quadcopter. The sensors will be leveled in respect to the quadcopter ground position. For example, the quadcopter determines its’ height through finding the difference in distance between the start and the current position. Therefore, reset feature is to zero the starting distance. As a result, the sensor on quadcopter will have a more accurate measurement of relative location from the starting off platform.

***Shutdown***
Shutdown function turns off the quadcopter and make it cease any type of transmission in the quadcopter system. It will power off the quadcopter device itself.

**System Functional Specification**

Upon power on the quadcopter, the system will stay in IDLE mode for command. The command will be received in Raspberry Pi through Secure Shell (SSH) protocol from a remote computer. The system will have a closed loop feedback control for the stability and flight control of the quadcopter from the input of the collected data from sensors. If the capture command is entered, the Pi camera will respond to the command accordingly.

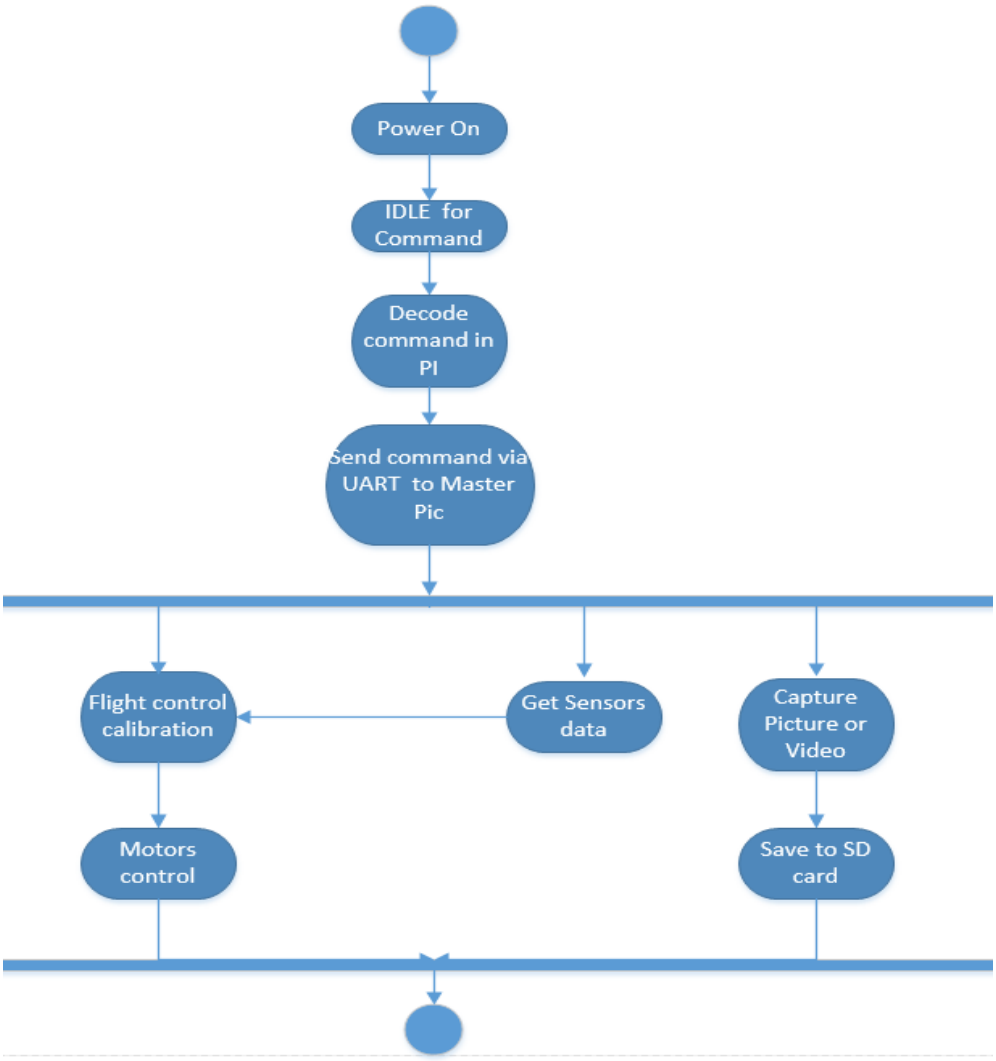



**Figure 3.20 Functional decomposition for Quadcopter System**

**Operating Specification**

Temperature Range: 20-50 C° typical, 90 max.
Humidity up to 90% RH non-condensing
Battery charging 120 – 240 VAC 50 Hz, 60 Hz, 400 Hz, 15 VDC
The system shall operate for a minimum of 15 mins on a fully charged battery
The system time base shall meet the following specifications

Aging Rate
90 day < 3 x 10-8
6 month < 6 x 10-7
1 year < 25 x 10-6

**Reliability and Safety Specification**

The quadcopter should have full blade protection for propellers and an emergency landing plan. The quadcopter should be compliant with the AMA National Model Aircraft Safety Code. The use of quadcopter should be compliant to state with federal privacy laws regulations.

**Individual Modules:**

**Power Module**

The power module mainly consisted the 2.25A Step Down Input Adjustable Buck Switching Regulator to regulate the power of our entire quadcopter. The Switching Regulator used is TI's PTH08080WAH. The purpose of the power module is to regulate the voltage from the input 11.1 V from the LiPo battery to 5 V that drives other modules on board such as the raspberry pi, sensor card, and flight controller. The power card supports up to 3 regulated output.

In order to protect the overall powerboard and rest of the electronics on board from unexpected surge of current or high voltage, the power module also includes a fuse before the input the 11.1V to the switching regulator. The power module also has a toggle switch (TE Connectivity's MTA106DPC) that supports up to 6A and 125V to power on or power off all the electronics easily.

- Input: 11.1 V from LiPo Battery
- Output: 3 5V ports to power other modules on board
- Side Effects: None noted
- Constraints for Switching Regulator:
    - Vin(min): 4.5V | Vin(max): 18V
    - Vout(min):0.9V | Vout(max): 5.5 [our application sets Vout as 5V]
    - Iout(max) :2.25A
    - Temperature Range: -40 to 85 C
- Error Handling:
    - If an unexpected occurrence of current or voltage is too high, the fuse will be blown to protect the switching regulators from any damage.

**Flight Controller**

Flight controller is the "brain" of the quadcopter packed in one PIC18F22k25. It processes the command from the client side via Raspberry Pi which connects to PIC through RS232. It controls the four motors on quadcopter; therefore, controlling the entire maneuver of the quadcopter. In which, flight controller also includes the stabilizer that balances the quadcopter when it is up in the air. In order to stabilize the quadcopter, it needs sensor reading from the sensor pack: gyroscope, accelerometer, and compass mounted on the quadcopter.

The flight controller is going to adjust each motor's speed in order to balance itself. The flight controller take inputs from the sensor to adjust it's yaw, pitch, and roll with a stabilizer derived from a PD controller. The output of the PD controller is sent out to each motor with adjusted throttle to fix the quadcopter back to its desired position.

Inside the PIC, there are several module, each responsible for an aspect of the flight controller. Controller: initializes the intrinsic of the quadcopter as well as processing each user command sent via the Raspberry Pi to execute the expected action with the given user input such as increasing throttle, crash landing, and etc. The whole controller function and what it controls are detailed in the software implementation.
Intra_Com: communicates with flight controller's slave, the sensor card, through $I^2C$ to get data for yaw, pitch and roll for stabilizer.
Stabilizer: Computes the output to the motors based on the error from sensor reading and expected position. Stabilizes the roll, pitch, yaw and altitude.
Motor drive: sets the PWM needed for ESC that drives the motor. Greater PWM increases the speed of motor. Sets PWM based on output from stabilizer after normalization.
Scheduler: Runs the entire flight controller. Sets timer interrupt to read sensor, stabilizer routine, flight controller routine, UART routine, and etc.
UART: Communication with Raspberry Pi to receive user commands to send into the Controller.

- Inputs: UART command, $I^2C$ Sensor data
- Outputs: PWM to ESC
- Side Effects:
- Timing Constraints: Scheduler runs at 16 Hz
- Error Handling:
    - If the quadcopter stability is out of control, there is a crash landing control that stops all the motor spinning

**Sensor**

We implemented 2 sensors chip for this project, GY80 and GY512
Gy80 is 9 axis accelerometer, gyroscope and compass. L3G4200D is an Angular Rate Sensor, it measures the speed the device rotates (radians per second). ADXL345 is an Accelerometer (in g's) HMC5883L measure the magnetic field related to each of the 3 axis (X, Y, Z).

Unfortunately, Gy80 was burnt, so we have to switch to GY512. GY512 includes IMU6050 which consists of 3 axis Gyroscope and 3 axis Accelerometer.

The input of the system is from the raw data from the sensors while the output is the raw pitch and yaw. The data will be sent through I2C to flight Controller PIC to process the data to calculate PID controller. The sampling rate for our sensor card is about 16 Hz. Due to limitation of PIC, 16 Hz is the fasted we can go with the scheduler. With the slower sampling rate, there would be gyro drift. Therefore, we measure pitch and roll by just using Accelerometer with a digital low pass filter. And we used the compass to measure yaw.

Pseudo Code:

Roll = arctangent of X acceleration in g's and z acceleration in g's multiply by the 180 divide by pi (to convert it degree)

pitch = arctangent of Y acceleration in g's and Z acceleration in g's multiply by the 180 divide by pi ( to convert it degree)

yaw = arctangent of x compass and y compass * 180 / pi + 180

## SOFTWARE IMPLEMENTATION

### Flight Controller

For flight controller, there are many modules that controls different aspects of flight control. This is a huge module since as mentioned before, this is the "brain" of the quadcopter. Here are the details block diagrams of each modules' function as well as the flow of these functions.

The block diagram below details the controller module. There are two modes, auto pilot and manual mode. Autopilot mode relies on the stabilizer to balances itself and lets the autopilot perform it's maneuver. On the other hand, there are also the manual control function that provide the user with more control over the quadcopter's movement by controlling the roll, pitch, throttle, and yaw offset.

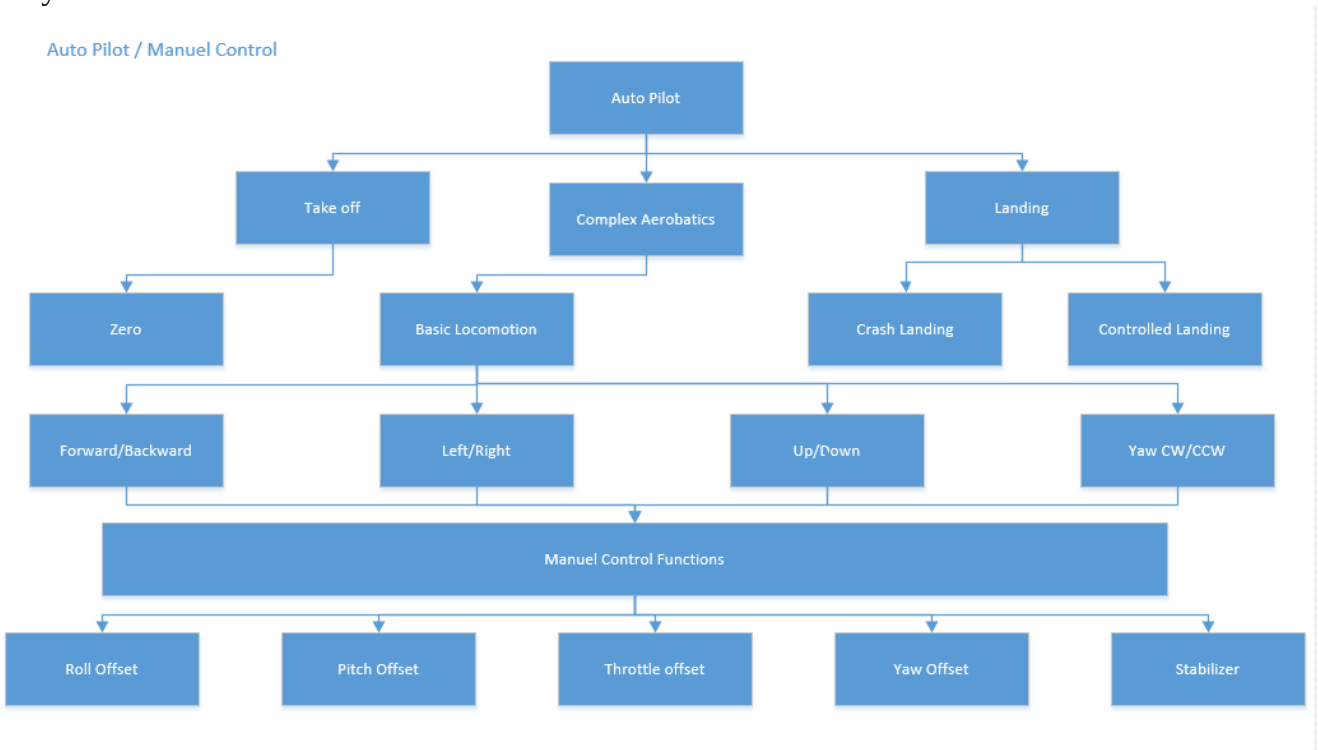



**Figure 3.21 Autopilot/Manual Control Block Diagram**

The following block diagram specifies the stabilizer function inside the flight control. This is *the* crucial part for having a stable hovering. In order to hover, we need to control the roll, pitch, and yaw through our PD controller digital filter that outputs the adjusted throttle. Stabilizer calculates the needed output through updated error, which is the difference between the expected sensor readings to the measured sensor reading. The result of the stabilizer should be able to control the four motors to have a stable hovering state.

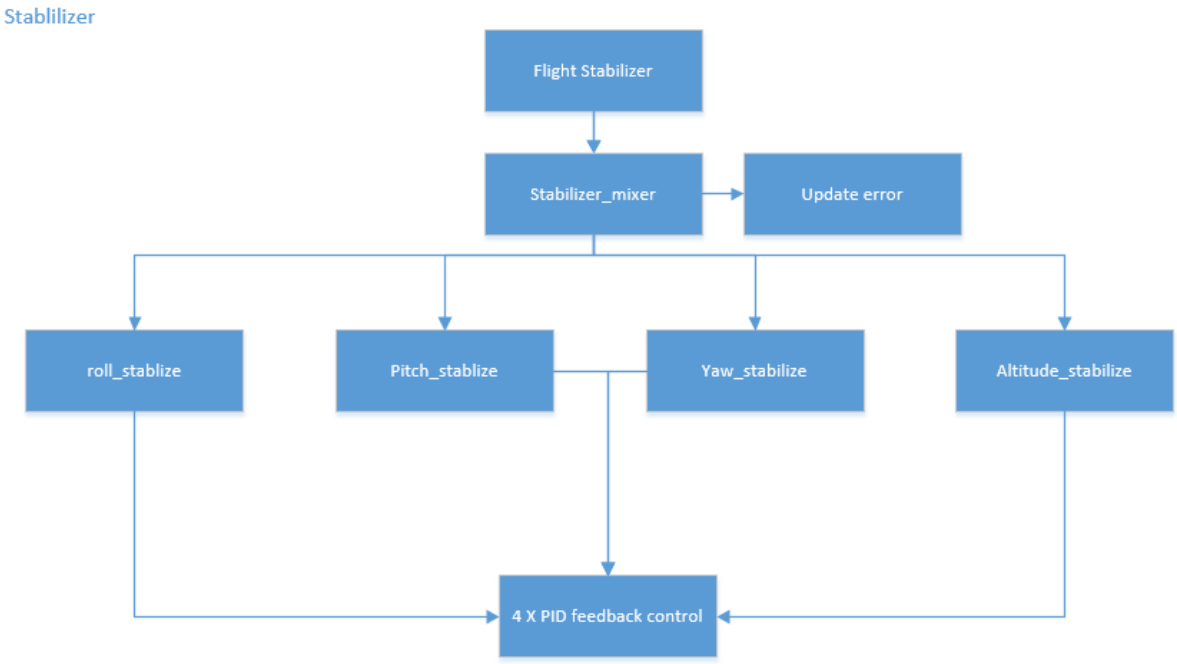



**Figure 3.22 Stabilizer Block Diagram**

## Raspberry Pi

The following block diagram specifies how we utilizes Raspberry Pi for our quadcopter project. We access Raspberry Pi through SSH (Secure Shell). Although as powerful as Raspberry Pi is, we only use Raspberry Pi for sending data through Wi-Fi and the picture taking capabilities. Raspberry Pi should be able to record video, take picture as well as communicate with the flight controller through serial communication for user commands.

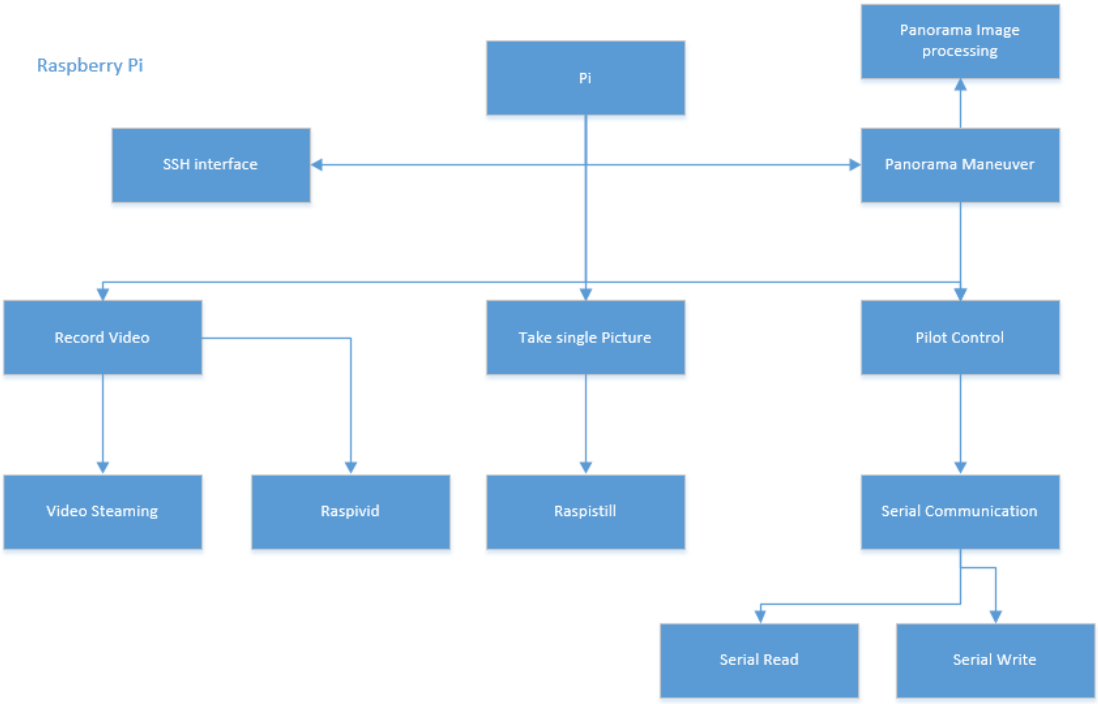



**Figure 3.23 Raspberry Pi Block Diagram**

**Sensor**

As seen as the block diagram below, our sensor module consists of reading from the sensors: accelerometer, compass, and gyroscope. These are read to the Sensor Unit PIC through $I^2C$ and the sensor unit has to fuse the datas through filter in order to send it back up to the flight controller for stability. The flow of the sensor unit and it's software components are shown as below:

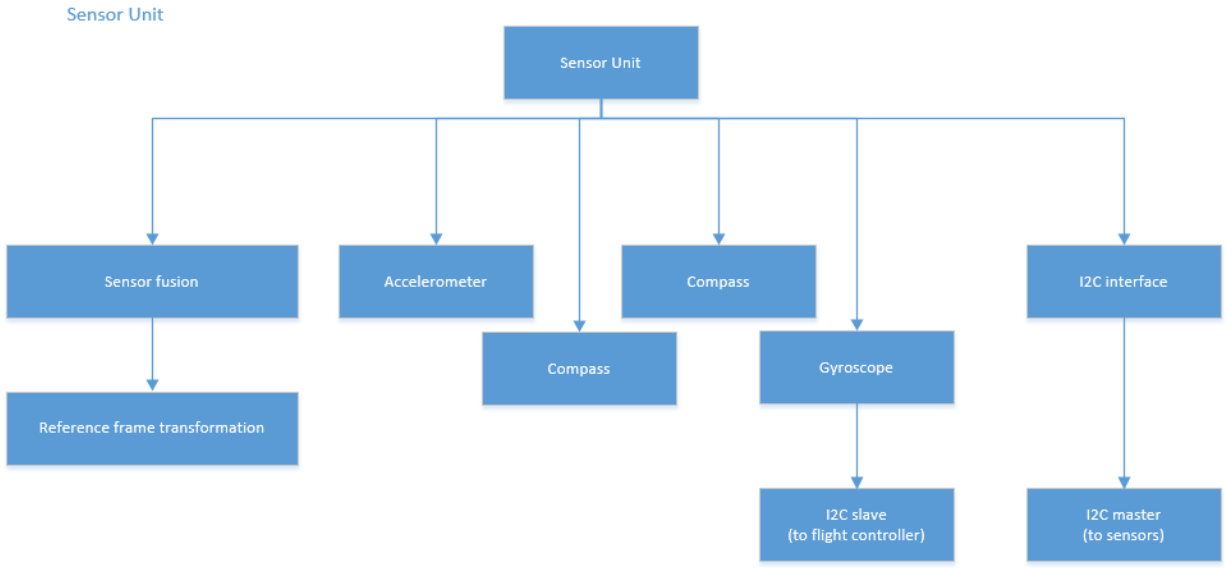



**Figure 3.24 Raspberry Pi Block Diagram**

**Sensor Setup for GY80**

**ADXL345**

```
//Put the ADXL345 into +/- 2G range by writing the value 0x01 to the DATA_FORMAT register.
 master_I2C_write_byte(ADXL345_ADDRESS, ADXL345_RA_DATA_FORMAT, 0b00001011);
//set data rate to be 50 Hz
 master_I2C_write_byte(ADXL345_ADDRESS, ADXL345_RA_FIFO_CTL, 0x00);
 master_I2C_write_byte(ADXL345_ADDRESS, ADXL345_RA_BW_RATE, 0x09);
 master_I2C_write_byte(ADXL345_ADDRESS, ADXL345_RA_POWER_CTL, 0x08);
```

**L3G4200D**

```
// set CTRL_REG1 in L3G4200D
 // bit 7-6 output data rate
 // bit 5-4 bandwidth selection
 // bit 3        power down enable
 // bit 2        Z axis enable
 // bit 1        Y axis enable
 // bit 0        X axis enable
```

```
   master_I2C_write_byte (L3G4200D_ADDRESS, L3G4200D_REG_CTRL_REG1,
0b01101111);

   // set CTRL_REG3 in L3G4200D
   // bit 7        INT1 pin interrupt enable
   // bit 6        Boot status available on INT1
   // bit 5        Interrupt active config on INT1
   // bit 4        Push - pull / Open drain for I2C
   // bit 3        Data ready on RDY/INT2
   // bit 2        FIFO watermark interrupt RDY/INT2
   // bit 1        FIFO overrun interrupt RDY/INT2
   // bit 0        FIFO empty interrupt RDY/INT2
   master_I2C_write_byte(L3G4200D_ADDRESS, L3G4200D_REG_CTRL_REG3,
0b00010000);
```

**HMC5883L**

```
   //15Hz
   master_I2C_write_byte(HMC5883L_ADDRESS, HMC5883L_RA_CONFIG_A, 0x70);
   //Gain = 5
   master_I2C_write_byte(HMC5883L_ADDRESS, HMC5883L_RA_CONFIG_B, 0xA0);
   //coninuous measurement
   master_I2C_write_byte(HMC5883L_ADDRESS, HMC5883L_RA_MODE, 0x00);
   //timer0_delay(300);
```

**Sensor Setup for GY512**

**IMU 6050**

```
   //Sets sample rate to 8000/1+7 = 1000Hz
   master_I2C_write_byte (MPU6050_ADDRESS, MPU6050_RA_SMPLRT_DIV, 0x07);
   //Disable FSync, 256Hz DLPF, DLPF_CFG =4
   master_I2C_write_byte (MPU6050_ADDRESS, MPU6050_RA_CONFIG, 0x00);
   //Disable gyro self-tests, scale of 500 degrees/s
   master_I2C_write_byte (MPU6050_ADDRESS, MPU6050_RA_GYRO_CONFIG,
0b00001000);
   //Disable accel self-tests, scale of +-2g, no DHPF
   master_I2C_write_byte (MPU6050_ADDRESS, MPU6050_RA_ACCEL_CONFIG, 0x00);
```

**Pseudo Code for calculating the sensor card**

roll = arctangent of X acceleration in g's and z acceleration in g's multiply by the 180 divide by pi ( to convert it degree)

pitch = arctangent of Y acceleration in g's and Z acceleration in g's multiply by the 180 divide by pi ( to convert it degree)

Yaw = arctangent of x compass and y compass * 180 / pi + 180

**Streaming**
We use Pi camera and mjpg streamer software to stream out pictures one by one. The rate of streaming can be adjusted those it is resource friendly.

After the installation the code below is used to start the Pi.

To start the streaming

$ mkdir /tmp/stream

$ raspistill --nopreview -w 640 -h 480 -q 5 -o /tmp/stream/pic.jpg -tl 100 -t 9999999 -th 0:0:0

&LD_LIBRARY_PATH=/usr/local/lib mjpg_streamer -i "input_file.so -f /tmp/stream -n pic.jpg"

-o "output_http.so -w /usr/local/www"

-t is the period of streaming. 9999999- or 0 will take the pictures forever.

The streaming can be view from
http://<IP-address>:8080

## HARDWARE IMPLEMENTATION

There are three main components to the hardware of our quadcopter. The first one a power board that regulates the voltage of the entire quadcopter. The second one being the flight controller PIC that controls the ESCs and motors. Lastly, there is also a sensor board that has the sensor packs and the PIC that reads and fuses the data.

### Power Card

For the power card, the hardware consists of the switching regulators, capacitors, resistor, switches, fuse, and fuse holder. The switching regulator used is PTH08080W from TI that can output up to 2.5A and regulate voltage range from 0.9 to 5.5 V. The capacitors are 100μF electrolytic capacitors. The switch is TE Connectivity's MTA106DPC toggle switch. The fuse used is 0218.040HXP from LittelFuse that supports up to 2.5A and 250 VAC. The fuse holder to hold the fuse is a 22.5mm 5x20 fuse holder from LittelFuse as well (PN: 65600001009).

The power board connections are based on the switching regulator PTH08080W. From the regulator's datasheet, it specifies the needed external parts to make the regulator function as seen below. The connections and capacitor values are connected accordingly.

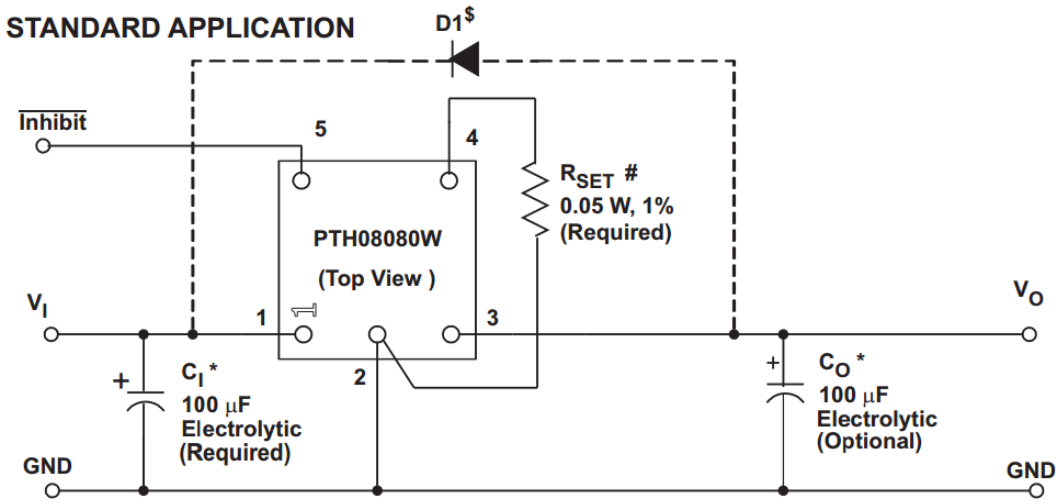



**Figure 3.25 Switching Regulator Connection**

The table below is also from PTH08080W's datasheet that specifies what the resistance value (RSET#) should be in order to obtain the desired regulated output. For our application, we want a 5V output. As a result, our RSET# is set to 348 Ω.

| $V_O$ (Required) | $R_{set}$ (Standard Value) | $V_O$ (Actual) |
|---|---|---|
| 5 V(1) | 348 Ω | 5.010 V |
| 3.3 V | 1.87 kΩ | 3.315 V |
| 2.5 V | 3.74 kΩ | 2.503 V |
| 2 V | 6.19 kΩ | 2.012 V |
| 1.8 V | 8.06 kΩ | 1.802 V |
| 1.5 V(2) | 13.0 kΩ | 1.501 V |
| 1.2 V(2) | 27.4 kΩ | 1.205 V |
| 1 V(2) | 86.6 kΩ | 1.001 V |
| 0.9 V(2) | Open | 0.9 V |

**Figure 3.26 Resistor values of various regulated output**

With the fuse and switches, it adds extra layer of protection and robustness to the power board. The overall connection block diagram can be seen as below with detailed switching regulator pin connection described in the PTH08080W's datasheet in Figure 3.25.

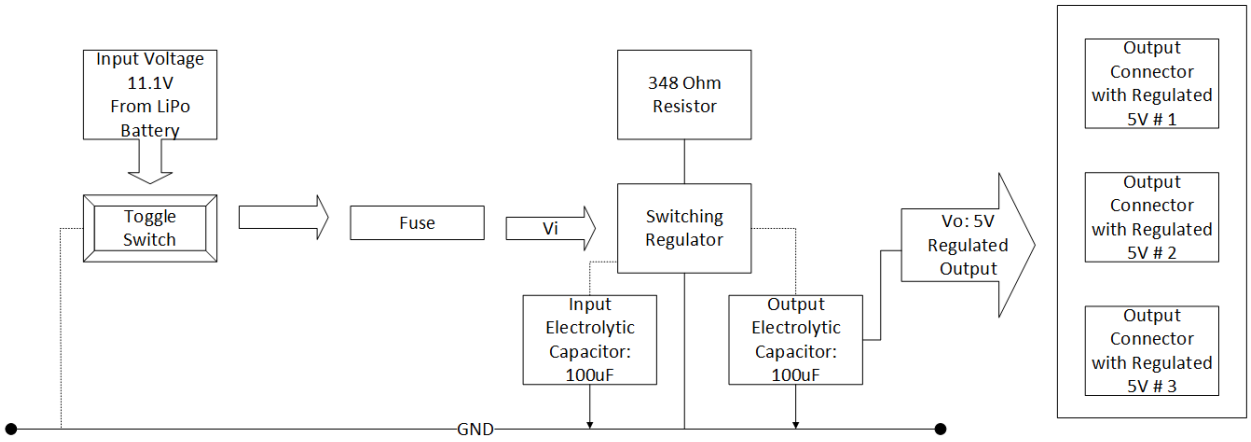



**Figure 3.27 Overall Connection for Power Board**

The actual power board hardware is as below:

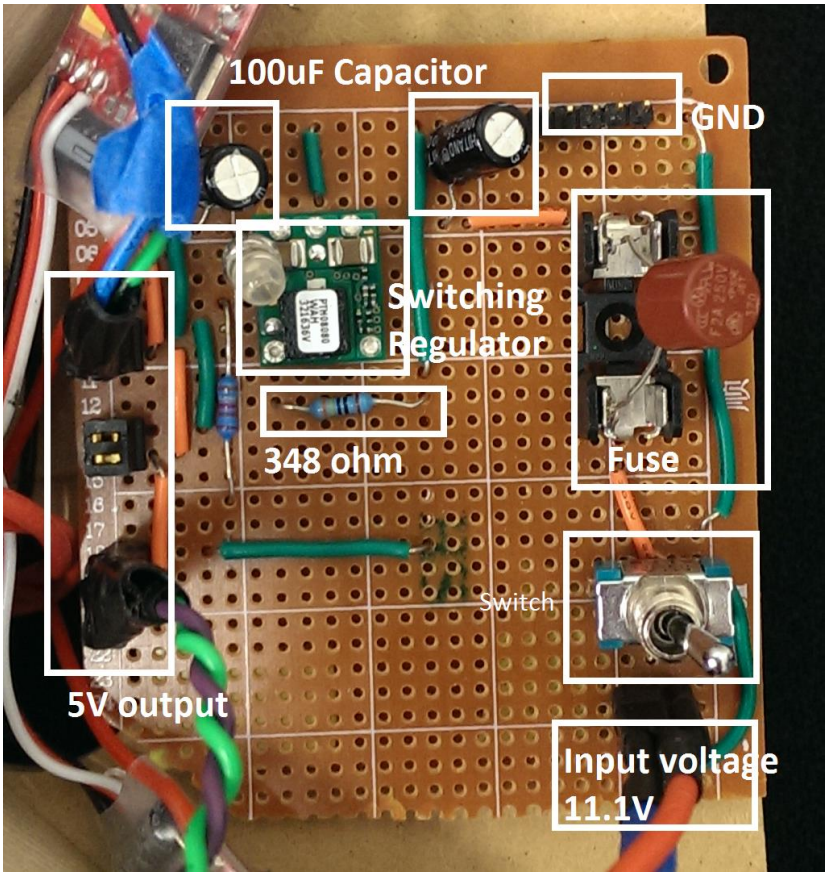



**Figure 3.28 Quadcopter Power board**

**Flight Controller**

Flight controller consists of PIC18F22K25 that moderates the entire quadcopter flight. It is connected to the ESCs and Motors. Our quadcopter uses a 10A flyFun ESC (Electronic Speed Control). The ESCs takes in 50Hz PWM that controls the speed of the motor. To set the speed, vary the duty cycle and the ESC will output a three phase signal to our SunnySky brushless DC motor. The PWM is output by the PIC.

For the connection of the PIC, please see the below diagram of circuit connection.

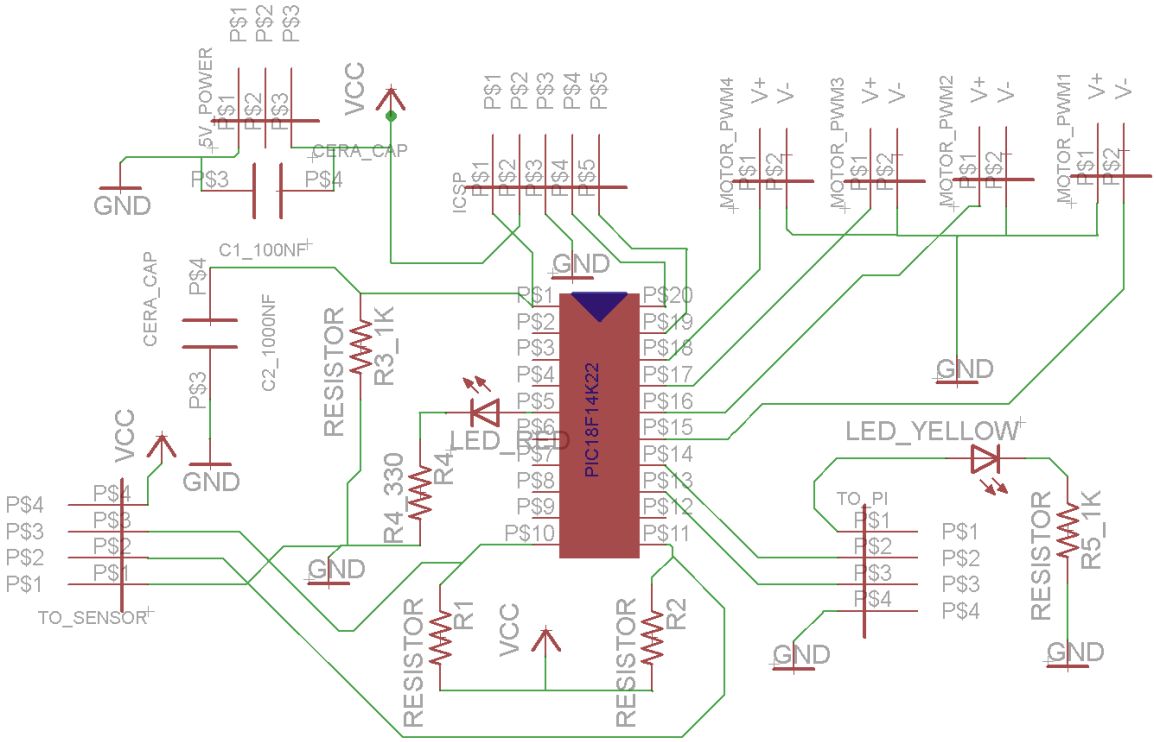



**Figure 3.29 Flight Controller Circuit Diagram**

This is the actual flight controller board for our quadcopter which has the connectors for the 4 ESCs on the left and serial communication to the Raspberry Pi with the white connector on the right.

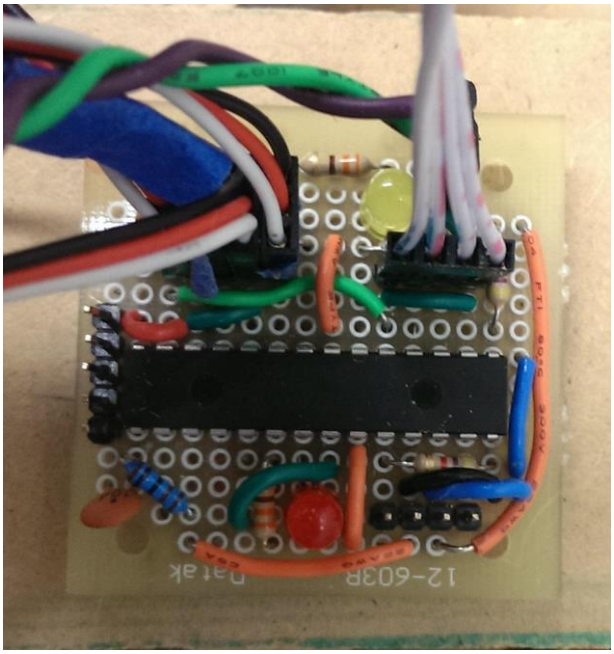


**Figure 3.30 Flight Controller Board**

GY521 contain IMU6050 with 3 axis Accelerometer and 3 axis Gyroscope while HMC5883L contains 3 axis compass. The slave pic is used to read in the raw data with 16 HZ frequency through I2C.  In the slave PIC, the data will be scaled and filter out to get appropriate raw, pitch and roll. The filtered data will be sent through I2C to the master PIC to calculate the PID controller. After computation in the master PIC, the speed of the motor will be determined and sent to PORTB.

The two LEDS are used to check if the schedulers are still working by turning off the LEDS every time it goes to the scheduler.
Sensor card circuit Diagram

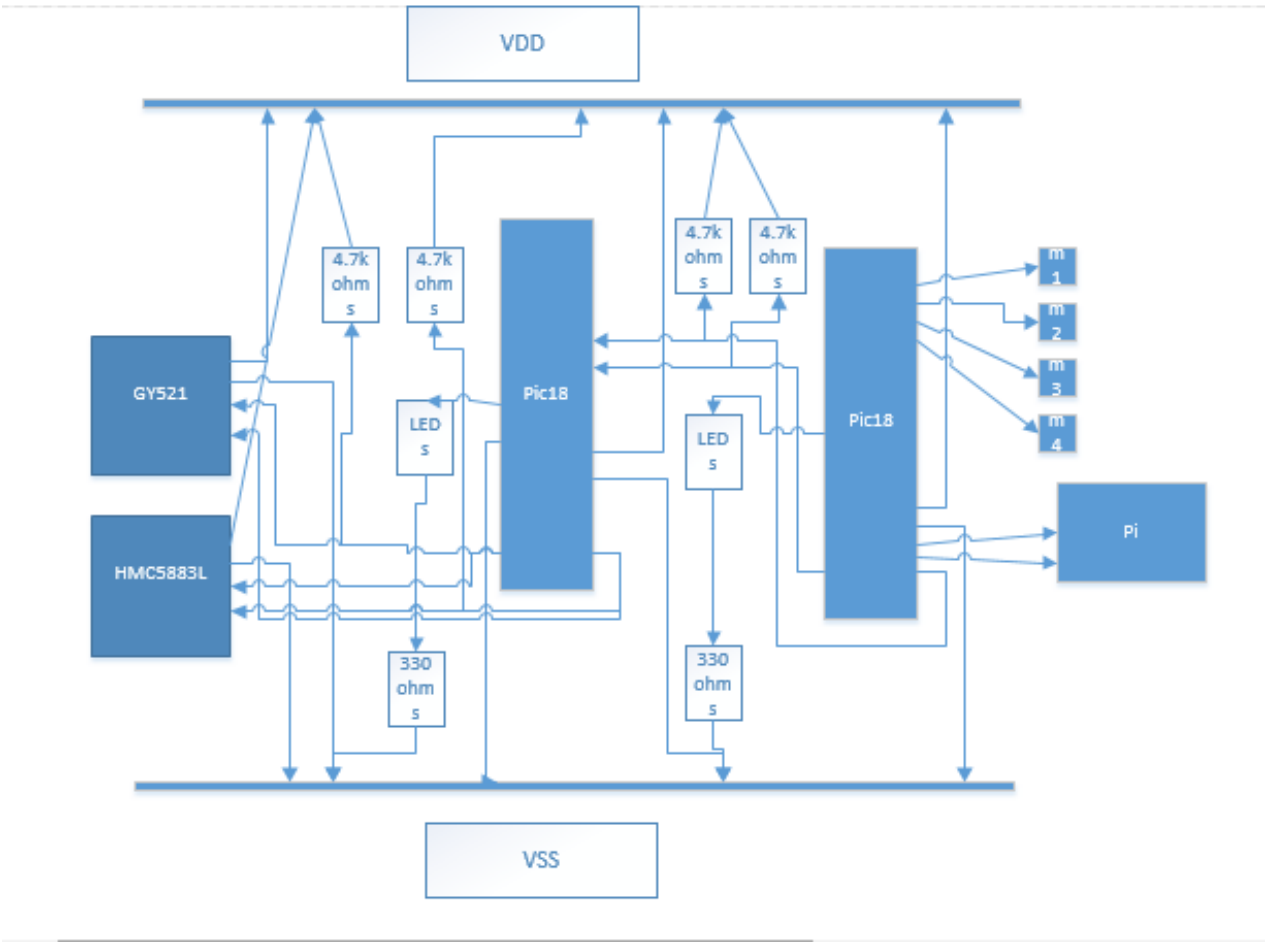



**Figure 3.31 Sensor Board and Flight Controller Connection**

This is the actual sensor board connection between the PIC and the sensor packs.

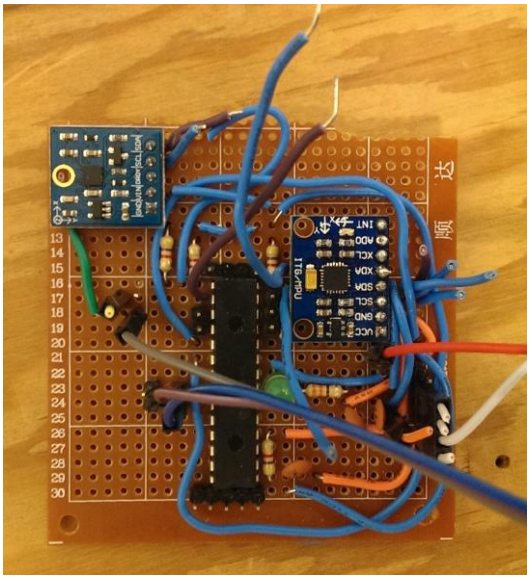


**Figure 3.32 Sensor Board**

**TEST PLAN**

Overall summary of what needs to be tested to ensure that your design meets the original requirements, 2-3 paragraphs maximum unless specified otherwise
Present your plan to prove that your circuit works as specified. How can you prove that circuit you designed meets all the specifications and extra features?
This is an annotated description of what is to be tested and the test limits
Note, this does not specify test implementation...this is ***what*** to do, not ***how***
In addition, you should also consider testing the "don't care" terms, the false conditions, and the boundary limits of the hardware elements of your circuit as well as any software inputs.

**Sensors Card**

IMU6050
Environment:

-Sets sample rate to 8000/1+7 = 1000Hz
-Disable FSync, 256Hz DLPF, DLPF_CFG =4
-Disable gyro self-tests, scale of 500 degrees/s
-Disable accel self-tests, scale of +-2g, no DHPF

The same environment setup is used to test for these four tests

1. Test for the raw sensor data to make sure the lower and higher bits registers are read accordingly.

2. Test for the scaled output according to the set-up of the sensors

3. Test for Sensors data after the fusions (roll, pitch, yaw)

4. Test if the data is sent correctly to Flight Controller PIC

**Streaming of the pi and web interface**
Testing the result of the streaming.

**TEST SPECIFICATION**

**Sensors Card**

Verify that the raw sensor data are accurate
Verify that the scaled output are right according to the set-up of the sensitivity and scale
Verify that roll, pitch, yaw are right after the filtering and calculation
Verify that the data through the I2C are right after the increasing the resolution.

Annotated description of what is to be tested and the test limits. This specification quantifies inputs, outputs, and constraints on the system. That is, it provides specific values for each.
Note, this does not specify test implementation...this is what to do, not how to do it.

## TEST CASES

Annotated description of how your system to be tested against the test limits
Note, this does specify test implementation...this is not ***what*** to do, this is ***how*** to do it based upon the test specification.

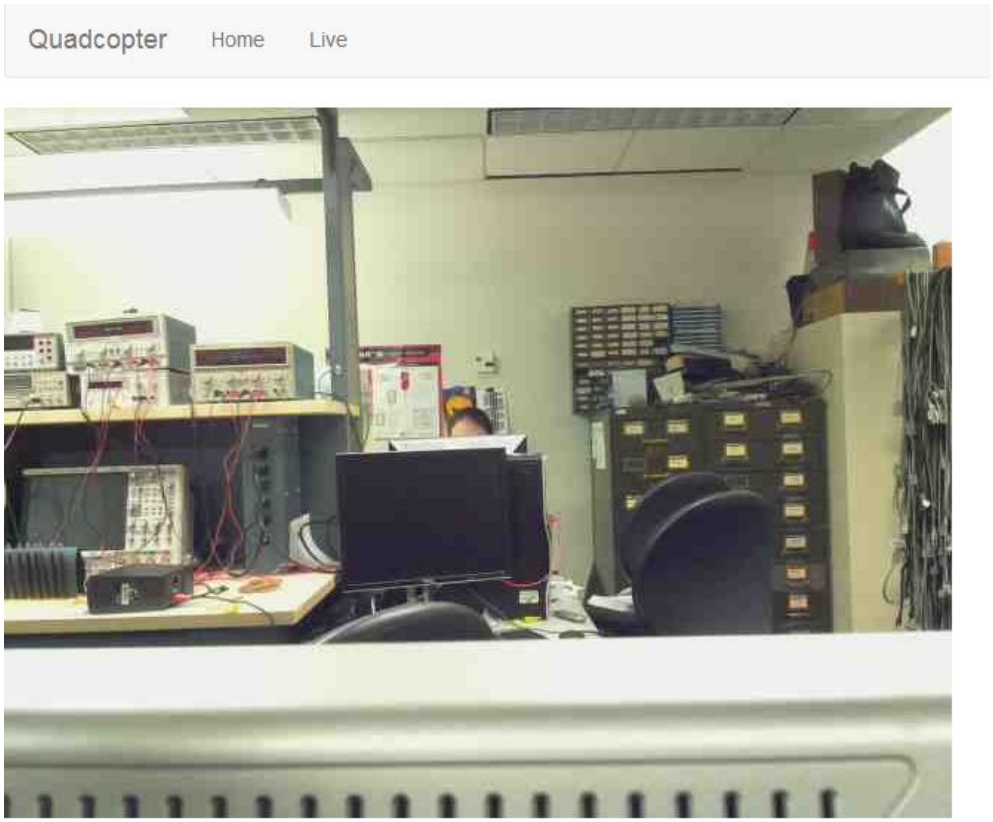
## PRESENTATION, DISCUSSION, AND ANALYSIS OF RESULTS

Based upon the execution of your design, present your results. Explain them and what was expected, and draw any conclusions (for example, did this prove your design worked).
In addition to a detailed discussion and analysis of your project and your results, you must include all the answers to all questions raised in the lab.

### Integration testing of the whole system

We tested the quadcopter with a metal stand in the beginning. After behavior was observed, we moved the test platform in the field. We a number of flight runs. We tested the stabilization the stabilizer can be viewed in YouTube.
https://www.youtube.com/watch?v=GHrgVgztJYM

The quadcopter tried to stabilize the itself by trying to offset the error of pitch, roll or raw with increasing and lowering the four motor‘s speed.




**Streaming of the Pi camera**

Web interface of our website that can be used to view the live stream from the pi camera

The streaming of  the pi works perfectly through Wi-Fi. The user can look at the stream of the pi camera as seen in the pictures above.

## ERROR ANALYSIS

Due to the complexity of UAV project, we ran into numerous problems during the design and development process. We managed to find solutions for most of these problems but still have problems ahead of us. Before the demo day we managed to get the quadcopter off the ground but we understand that flight is not a problem that could simply be answered by worked or not worked. The flight performance can be described by a number of factors and to get a flight performance that is commercially acceptable requires long-term development and testing. For this section we list the problems we encountered during development that we found a solution for, problems that we adopt a cheap solution for, and that problems that we have not yet managed to overcome. We listed these problems separately in the following sections.

**Problems Fixed:**

**Timing of the Raspberry Pi**

The pi needs to constantly listen to asynchronous inputs from the user and the flight controller, provide a graphical interface and provide streaming service via data, and interface the camera. Multitasking in the pi is managed by the linux operating system. To interface the pi successfully with the flight controller, which is a real-time system successfully, we need to emulate the pi into a real-time system as much as possible. To achieve the above goal, we minimized the process running inside pi and used python multithreading programming. We created three threads inside the pi that are constantly running when we started quad_GUI.py, namely, the serial thread, the console thread and the keyboard thread. It turned out the pi could successfully perform all its expected functionality in the end.

**Feasibility of the Project**

The initial design of the quadcopter involved a large amount of aerodynamic calculation to ensure that the motors could provide enough lift for the quadcopter to hover, and to provide enough maneuverability. The motor torque and speed, the current and capacity of the battery, the shape and strength of the propellers, the current limit of the speed controllers as well as the overall weight all require extensive design choices. When we were researching on previous quadcopter projects in this class, we found that this part was largely ignored and some resulted in inappropriate selection of parts. To overcome this, we conducted extensive research on helicopter theories, pinpointing the momentum theory, the blade element theory, and stepper motor analysis as our theoretical base and worked out part selection. We verified our design using ecalc, a RC calculator in the web. It turned out the quadcopter is able to lift and maneuver while sustaining the battery for a long time in our flight test.

**Frame Overweight**

At one point during the design, we weighted individual parts and collected the following data. From the initial calculation, we have a maximum load of 1550g but our measurement goes to 1755.7g. To reduce the weight, we removed the top panel that was used to enclose the battery and the motor controllers and seal the battery. We also drilled holes on the quadcopter frame randomly to further reduce the weight. In the end we successfully reduced the weight of the quadcopter to weight below 1550g.

Quadcopter weight

| Mechanical + Power | Unit name | # of units | Unit weight(g) | Total Weight(g) |
|---|---|---|---|---|
| | ESC | 4 | 9 | 36 |

| | | | | |
|---|---|---|---|---|
| | Bottom plate(Motor + propeller + bottom frame + misc screws) | 1 | 900 | 900 |
| | Top plate | 1 | 152.4 | 152.4 |
| | Aluminium Shield | 4 | 10 | 40 |
| | Battery | 1 | 414.2 | 414.2 |
| Electronics | PI | 1 | 61.5 | 61.5 |
| | PI cam | 1 | 5.2 | 5.2 |
| | Power card | 1 | 26.4 | 26.4 |
| | Sensor card | 1 | 50 | 50 |
| | Control card | 1 | 50 | 50 |
| | Misc wires | 1 | 20 | 20 |
| | Model weight (without drive) | | | 1755.7 |

**Propeller Failers**

During the development we had some problems with the propellers. At the initial state, we ordered 4 counterclockwise APC10x4.7 propellers. We sadly ignored the fact that propellers, while rotating, induce a force horizontally forcing the quadcopter to rotate. As a result we need to use two counterclockwise propellers and two clockwise propellers to perform successful yaw control. After reordering propellers of the right rotation direction, we realized our new propellers have a really low thrust rating, which fail after the rotation speed reaches 3000 rpm. In the end we ordered composite propeller of acceptable strength and direction and no errors of propellers were reported after.

**C18 I2C Library Function Problem**

During I2C development we realized that the I2C functions in the C18 library has unpredictable behaviors, especially for our particular throughput, which is really large between two PICs.

Fusion algorithm: Kalman filter/Complementary filter

Sensor fusion problem is the problem of fusing gyroscope, accelerometer and compass for better angular motion sensing. Since each sensor has significant flaws, e.x, gyro drift, raw motion data from the sensors not to be trusted. The sensor fusion algorithm is one of the biggest design challenge for this project and is an active research area. Even though having an inspiration for trying, implementing a kalman filter is much beyond our capabilities. We implemented a complementary but despite being simple, its effect was overwhelming. In the end, we decided to

use the gyroscope and the accelerometer only with processing. As a result, despite our effort to calibrate and filtering the sensor data, we likely got very questionable sensor outputs .

**PID Controller Causality Issue**

After developing the PID controller we simulated them in matlab. The output was beyond our wildest expectation. We verified our mathematical deduction and did not locate any errors. In the end, we found the problem was that the derivative term in the PID controllers, although was able to be written into code, was not physically possible to implement in real life. As a result we added an pole into the derivative term make the number of system poles equal to system zeros. It turned out the PID controller was able to produce much reliable outputs in the end.

**Z Axis Nonlinearity**

$$\ddot{Z} = -g + (\cos\theta\cos\phi)\frac{U_1}{m}$$

In the initial design stage we include the altitude controller as one of the PID controllers in the stabilizer. However, during the PID controller design we identified the nonlinear problem. As part of the physics model, the Z axis equation of motion is in fact a non-linear system. Due to the presence of gravity g, it is not possible to write the transfer function for the Z axis motion from the equation above. Closing a loop with such nonlinear plant with a linear PID controller would produce unpredictable results.

Due to lack of information on nonlinear control, the group decided to avoid the PID control by replacing the altitude controller with feedforward control. That is, the user now have direct control over the throttle instead of choosing the altitude they want. The throttle value held by the user is offset by the value of gravity. Since operating directly on throttle instead of altitude is acceptable for most users and some autopilot programs, this solution does not reduce the functionality of our design by too much. On the other hand, the benefits of adopting direct control over throttle is less dependency over sensors. The quadcopter can now operate without a level sensor and a temperature sensor.

**I2C Slave Interrupts**

Due to the frequency of communication, the microcontroller in the slave side was frequently interrupted by the master, as a result, some sensor fusion algorithm relying heavily on timing became unstable. Since in the end we remove the usage of gyroscope due to a large gyro drift presence, master interrupts did not become a problem.

**PID controller PD/PID problem**

The integral term of PID controllers, due to integration operations, could render numerical overflow in the PID controllers. While tuning the PID controllers, we realized that the a good PID controller for our plant usually have a really small gain for the integral term. We came to realize that a PD controller, which is a PID controller without the integral term, might do the

stabilization job equally well. From that point on we implemented our PID controller as PD controllers and it turned out that they produced similar outputs to the quadcopter both in simulation and real-life flight testing.

**Streaming Problem**

We used to worry that the video streaming might take too much CPU time from the raspberry pi. However, as our streaming tool is really light-weight, streaming can be executed along with our interfaces without any problem.

**Sampling Rate**

**Motor Saturation**

**Slow UART Problem**

The UART in our PIC18 microcontrollers, which provide access between the microcontroller the raspberry pi, has a tradeoff between error rate and communication speed. Since it is through this communication line that user input gets interpreted by the flight controller, and one single mistake in this data could resulted in disastrous output, we conservatively chose the most reliable baud rate, which is unfortunately the most time-consuming. The consequences of this design choice is a really complicated scheduling problem. In the end we carefully allocated our scheduler resources so that UART can take its time.

| FOSC | Baud_rate | number | actual | actual baud rate | error |
|---|---|---|---|---|---|
| 64000000 | 9600 | 415.6667 | 416 | 9592.326139 | 0.08% |
| | 19200 | 207.3333 | 207 | 19230.76923 | 0.16% |
| | 38400 | 103.1667 | 104 | 38095.2381 | 0.79% |
| | 57600 | 68.44444 | 68 | 57971.01449 | 0.64% |
| | 115200 | 33.72222 | 34 | 114285.7143 | 0.79% |
| | 230400 | 16.36111 | 16 | 235294.1176 | 2.12% |

**Problems Not Fixed:**

**Quadcopter Overweight**

Even Though trying our best to reduce the quadcopter weight, the quadcopter is still overweight, that is, the measured weight is really close to our computed weight limit. The impact of being overweight is as significant as being overweight as a human. One of the major consequence is the that quadcopter need to operate at 85% throttle to hover, which means it is really easy for the motors to saturate. In case of a

really large external distanburce, the quadcopter might not respond to it well. On the other hand, the quadcopter has almost no loads available, which reduces its power to carry heavier load like a large and decent camera, a machine gun(just kidding), or pizza/drinks for delivery. Our answer to this problem is, as Professor Peckol suggested, punch more holes to the frame, making the frame smaller or with lighter materials.

**Fusion Algorithm & Gyro drift**

Sensor fusion problem is the problem of fusing gyroscope, accelerometer and compass for better angular motion sensing. Since each sensor has significant flaws, e.x, gyro drift, raw motion data from the sensors not to be trusted. The sensor fusion algorithm is one of the biggest design challenge for this project and is currently an active research area. Even though having an inspiration for trying, implementing a kalman filter is much beyond our capabilities. We learned about a lite solution, namely the complementary filter which high pass the accelerometer output and low pass the gyroscope output before adding them up with a carefully selected weights. We implemented the complimentary but despite being simple, its effect was underwhelming. In the end, we decided to use the gyroscope and the accelerometer only with processing since the gyroscope drift still leads our data to unknown domain really fast after its turned on. As a result, despite our effort to calibrate and filtering the sensor data, we likely got very questionable sensor outputs. Some groups reported to successfully balance their quadcopters with the accelerometer only so we assume our problems are not the sensor problem alone.

**Constant Identification Issues**

(m(scales either accurate but with limited capacity and or inaccurate but with enough capacity))

**Smoked Sensors**

After successfully acquiring data from the sensor breakout board GY80 for 2 weeks, the sensor card was burnt. The LED monitoring scheduling activities stopped blinking and no I2C output could be obtained. We isolated parts in the sensor card and located the problem to be a toasted voltage regulator in the GY80 PCB. We might have shorted the circuit by not insulating the connection of the bottom circuit board but the exact reason for the overcurrent is unknown. Apparently a fuse along does not provide enough circuit protection.

Not able to find surface mount replacement, we have to explore other options. We checked replacement options for GY80 and the items was out of stock. 3 days before the demo, we purchased for some reasons, GY521 was burnt with the measured voltage of SCL of 1.6V while SDA was 3.3V. The sensors did not respond to the signal. We thought that there must be something wrong with the pull-up resistors as the SCL was lower than the usual voltage.

When gyroscope data changes faster than the sampling frequency, we will not detect it, and the integral approximation will be incorrect. This error is called gyro drift, as it increases in time. It results in the sensor reading not returning to 0 at the rest position. For this, it is important that we choose a good sampling period. The drift that we experienced was too big to integrate it to our filter. This might be due to the slow sampling rate of 16Hz from the PIC. The idea sampling rate for the gyroscope would be from 100Hz to 200 Hz.

**Martyrs of Quadcopter Project**


**Figure Burned ESC**

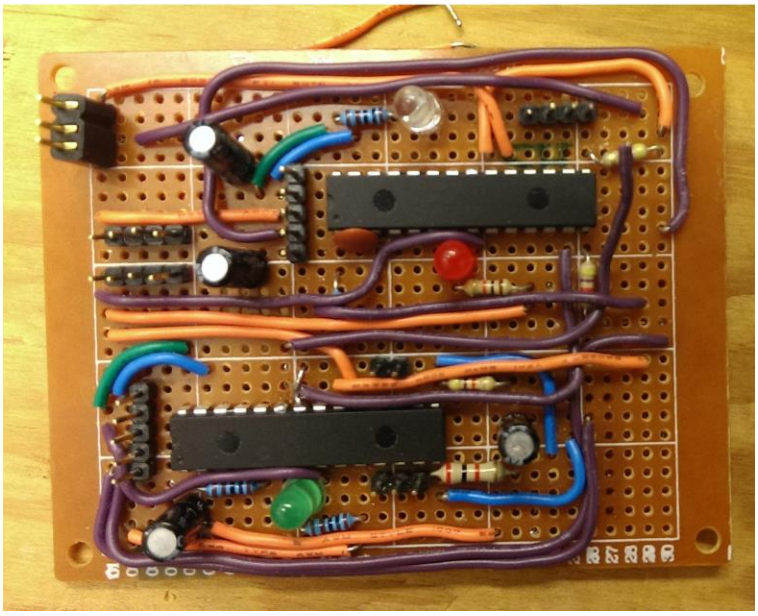


**Figure Sensor Card /Flight Controller 1.0**


**Figure Burned Pickit3**

## SUMMARY AND CONCLUSION

You should know these sections very well, no need to explain. Note, however, that they are two different sections. The summary is just that, a summary of your project. It should loosely mirror the abstract with a bit more detail. The conclusion concludes the report, potentially adds information that is often outside the main thrust of the report, and may offer suggestions or recommendations about the project.

## APPENDICES

Matlab simulation code:

PID_tunner.m

```
% PID tuner controller for the quadcopter
% YAW

clear all
clc

syms s
%constants
b = 0.0000538; %na
d = 0.0000011; %na
l = 0.32;  %a
m = 1.6;
IXX = 0.1296;
IYY = 0.1296;
IZZ = 0.2272;
hl = 17.64;
```

```
al = 20;

% choose pitch yaw or roll
I = IZZ;

% plant definition
P_deno  = sym2poly(I*s^2);
P_num = 1;
P = tf(P_num, P_deno);

% sensor def
H = 1;

Kp = 12.306;
Ki = 0;
Kd = 6.015;

% controller type thing
rho = 10;
rho_deno = sym2poly(rho + s);
P_num = 1;
that_thing = tf(rho, rho_deno);

C = pid(Kp, Ki, Kd);

T = feedback(C*P, H);

% closed loop reponse to the PID controller
%step(T)

%plant response
%step(P)

% open up the tuner
pidtool(P)


Difference Equations Response
% Yaw

close all

b = 0.0000538; %na
d = 0.0000011; %na
l = 0.32;  %a
m = 1.6;
IXX = 0.1296;
IYY = 0.1296;
IZZ = 0.2272;
hl = 17.64;
```

```
T =100;
lift_step = 10;

%error input
% no error
% vec_error = [0;0;0;0];

% all error
% vec_error = [-10.25;7.74;-55.555;0];

% roll error
%  vec_error = [5;0;0;0];

% pitch error
% vec_error = [0;7.74;0;0];

% yaw error
vec_error = [0;0;1;0];

temp1 = 1/4/b;
temp2 = 1/2/l/b;
temp3 = 1/4/d;

% inverse motion matrix
IMM = [temp1, 0, 0-temp2, 0-temp3;
   temp1, 0-temp2, 0, temp3;
   temp1, 0 , temp2, 0-temp3;
   temp1, temp2, 0, temp3];

% error input vector
error = [0.0;0.0;0.0;0.0];
al = 0;

% error data storage
e = [0,0,0,0;
   0,0,0,0];

f = zeros(2,4);
U_vecs = zeros(4,T);
M2_vecs = zeros(4,T);

for n = 1:1:T
   if (n < 40)
      if (mod(n,10) == 0)
         al = al + lift_step;
      end
   end

   % inject roll errors
   if (n == 60)
      error = vec_error;
```

```
    end


    e(1,1) = error(1);
    e(1,2) = error(2);
    e(1,3) = error(3);
    e(1,4) = error(4);

    % ROLL PITCH YAW PIDs
    f(1,1) = (1.055 * e(1,1) - 1.007 * e(2,1) + 0.7917*f(2,1));% roll
    f(1,2) = (1.055 * e(1,2) - 1.007 * e(2,2) + 0.7917*f(2,2));% pitch
    f(1,3) = (1.105 * e(1,3) - 1.07 * e(2,3) + 0.8535*f(2,3)); % yaw

    %f(1,3) = (12.19 * e(1,3) - 11.16 * e(2,3) + 0.04173*f(2,3));
    f(1,4) = al;

    U_vecs(1,n) = f(1,4);
    U_vecs(2,n) = f(1,1);
    U_vecs(3,n) = f(1,2);
    U_vecs(4,n) = f(1,3);

    % store prev data
    e(2,1) = e(1,1);
    e(2,2) = e(1,2);
    e(2,3) = e(1,3);
    e(2,4) = e(1,4);
    f(2,1) = f(1,1);
    f(2,2) = f(1,2);
    f(2,3) = f(1,3);
    f(2,4) = f(1,4);

    M2_vecs(:,n) = IMM*(U_vecs(:,n));
    % INV
end

figure;
subplot(4,1,1);
plot(1:1:T, U_vecs(1,:));
title('U1');
subplot(4,1,2);
plot(1:1:T, U_vecs(2,:));
title('U2');
subplot(4,1,3);
plot(1:1:T, U_vecs(3,:));
title('U3');
subplot(4,1,4);
plot(1:1:T, U_vecs(4,:));
title('U4');
figure(2);
subplot(4,1,1);
plot(1:1:T, M2_vecs(1,:));
```

```
title('omega1_square');
subplot(4,1,2);
plot(1:1:T, M2_vecs(2,:));
title('omega2_square');
subplot(4,1,3);
plot(1:1:T, M2_vecs(3,:));
title('omega3_square');
subplot(4,1,4);
plot(1:1:T, M2_vecs(4,:));
title('omega4_square');
```

Step response plotter

```
% yaw
syms s
%constants
b = 0.0000538; %na
d = 0.0000011; %na
l = 0.32;  %a
m = 1.6;
IXX = 0.1296;
IYY = 0.1296;
IZZ = 0.2272;
hl = 17.64;
al = 20;
Kd = 0.095491;
Kp = 0.39099;
Ki = 0;

sampling_rate = 0.0625; % sampling period
k = 4;

I = IZZ;

% PD controller
h = tf([Kd/k+Kd, Kp], [Kd/Kp/k, 1]);

hd = c2d(h, sampling_rate, 'tustin');

% plant definition
P_deno  = sym2poly(I*s^2);
P_num = 1;
P = tf(P_num, P_deno);

oc = P * h;

H = 1;

T = feedback(oc, H);
Tcf = feedback(h, H*P);

subplot(4,1,1);
```

```
% controller response
step(h, '-', hd, '--');
title('controller response')
subplot(4,1,2);
% open loop response
step(oc, '-');
title('open loop response')
subplot(4,1,3);
% closed loop response
step(T, '-')
title('close loop response')
subplot(4,1,4);
% control effort
step(Tcf, '-');
title('controller effort')
```

Source code

Streaming code

Live.html

```
<html xmlns="http://www.w3.org/1999/xhtml"><head>
<title>Live-Streamer</title>
<script type="text/javascript">

var imageNr = 0; // Serial number of current image
var finished = new Array(); // References to img objects which have finished downloading
var paused = false;

function createImageLayer() {
  var img = new Image();
  img.style.position = "absolute";
  img.style.zIndex = -1;
  img.onload = imageOnload;
  img.onclick = imageOnclick;
  img.src = "/?action=snapshot&n=" + (++imageNr);
  var webcam = document.getElementById("webcam");
  webcam.insertBefore(img, webcam.firstChild);
}

// Two layers are always present (except at the very beginning), to avoid flicker
function imageOnload() {
  this.style.zIndex = imageNr; // Image finished, bring to front!
  while (1 < finished.length) {
    var del = finished.shift(); // Delete old image(s) from document
    del.parentNode.removeChild(del);
  }
  finished.push(this);
  if (!paused) createImageLayer();
```

```
}

function imageOnclick() { // Clicking on the image will pause the stream
  paused = !paused;
  if (!paused) createImageLayer();
}

</script>
<link rel="stylesheet"
href="https://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">

            <!-- our own stylesheet for overrides -->
<link rel="stylesheet" href="css/styles.css">


</head>

<body onload="createImageLayer();">

<div class="container">
                        <nav class="navbar navbar-default" role="navigation">
                            <div class="navbar-header">
                                <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".pawnee-navbar-menu">
                                    <span class="sr-only">Toggle Navigation</span>
                                    <span class="icon-bar"></span>
                                    <span class="icon-bar"></span>
                                    <span class="icon-bar"></span>
                                </button>
                                <a class="navbar-brand" href="index.html">Quadcopter</a>
                            </div>

                            <div class="collapse navbar-collapse pawnee-navbar-menu">
                                <ul class="nav navbar-nav">
                                    <li class=""><a href="quad.html">Home</a></li>
                                    <li><a href="live.html">Live</a></li>
                                </ul>
                            </div>
                        </nav>

                        <div id="webcam"><img src="/?action=snapshot&amp;n=19"
style="position: absolute; z-index: -1;">
                        <img src="/?action=snapshot&amp;n=18" style="position: absolute; z-
index: 18;"><img src="/?action=snapshot&amp;n=17" style="position: absolute; z-index:
17;"><noscript>&lt;img src="/?action=snapshot" /&gt;</noscript></div>

</div>


</body>
```

```
                <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
        <script src="https://netdna.bootstrapcdn.com/bootstrap/3.0.0/js/bootstrap.min.js"></script>
</html>
```

index.html

```
<html><head>
            <meta charset="UTF-8">
            <meta name="description" content="Home Page">
            <meta name="author" content="your-net-id">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <link rel="shortcut icon" href="img/logo.png">
            <title>Quadcopter Cameraman</title>

            <!-- hot link to Bootstrap stylesheet -->
            <link rel="stylesheet"
href="https://netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css">

            <!-- our own stylesheet for overrides -->
            <link rel="stylesheet" href="css/styles.css">

        </head>
        <body style="">
                <div class="container">
                        <nav class="navbar navbar-default" role="navigation">
                           <div class="navbar-header">
                              <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".pawnee-navbar-menu">
                                 <span class="sr-only">Toggle Navigation</span>
                                 <span class="icon-bar"></span>
                                 <span class="icon-bar"></span>
                                 <span class="icon-bar"></span>
                              </button>
                              <a class="navbar-brand" href="index.html">Quadcopter</a>
                           </div>

                           <div class="collapse navbar-collapse pawnee-navbar-menu">
                              <ul class="nav navbar-nav">
                                 <li class=""><a href="quad.html">Home</a></li>
                                 <li><a href="live.html">Live</a></li>
                              </ul>
                           </div>
                        </nav>

                        <div id="carousel-example-generic" class="carousel slide">
                          <!-- Indicators -->
                          <ol class="carousel-indicators">
                            <li data-target="#carousel-example-generic" data-slide-to="0"
class="active"></li>
```

```
                        <li data-target="#carousel-example-generic" data-slide-to="1"
class=""></li>
                        <li data-target="#carousel-example-generic" data-slide-to="2"
class=""></li>
                        <li data-target="#carousel-example-generic" data-slide-to="3"
class=""></li>
                      </ol>
                                        <!-- slides -->
                  <div class="carousel-inner">
                    <!-- slide -->
                    <div class="item active">
                      <img src="img/Picture2.png" alt="Summer in the Park">
                      <div class="carousel-caption">
                        <p>Our Prototype</p>
                      </div>
                    </div>

                            <div class="item">
                      <img src="img/Picture1.png" alt="Autumn in our Parks">
                      <div class="carousel-caption">
                        <p>Our Team</p>
                      </div>
                    </div>

                    <div class="item">
                      <img src="img/Picture3.png" alt="Winter in our Parks">
                      <div class="carousel-caption">
                        <p>Design</p>
                      </div>
                    </div>

                     <div class="item">
                      <img src="img/Picture5.png" alt="Spring in our Parks">
                      <div class="carousel-caption">
                        <p>Control</p>
                      </div>
                    </div>
                        </div>
                         <!-- Controls -->
                         <a class="left carousel-control" href="#carousel-example-generic" data-
slide="prev">
                           <span class="icon-prev"></span>
                         </a>
                         <a class="right carousel-control" href="#carousel-example-generic"
data-slide="next">
                           <span class="icon-next"></span>
                         </a>
                        </div>


                    <div class="row">
```

```
                        <div class="col-md-4">
                            <h2><a href="#">Design</a></h2>
                                    <img src="img/design.png" alt="Design" class="col-pic">
                            <p>Our design is robust with cutomized embedded system design,
master and slave I2C pic communication bwtween sensors</p>
                        </div>
                        <div class="col-md-4">
                            <h2><a href="#">Team</a></h2>
                                    <img src="img/team.png" alt="team" class="col-pic">
                            <p>Our highly motivated team consists of Jian Ma, Rebecca Chu,
and Oat Liewsrisuk. All are senior in Electrical Engineering at the Univeristy of Washington</p>
                        </div>
                        <div class="col-md-4">
                            <h2><a href="#">Future Plan</a></h2>
                                    <img src="img/future.png" alt="Future plan" class="col-
pic">
                            <p>Our Future plan is to create a highly robust quadcopter which
can be used as a cameraman or deliver guy</p>
                        </div>
                    </div> <!-- .row -->
            </div>

            <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>
    <script src="https://netdna.bootstrapcdn.com/bootstrap/3.0.0/js/bootstrap.min.js"></script>
     <!-- auto-play the carousel -->
        <script>
            $('.carousel').carousel();
        </script>

</body></html>
```

Sensors Card

MPU_6050.c

```
#include "MPU_6050.h"

#include <math.h>

#define GYRO_SENSITIVITY 65.5
#define GYRO_XOUT_OFFSET 4
#define GYRO_YOUT_OFFSET -11
#define GYRO_ZOUT_OFFSET 11
void Setup_MPU6050(void)
{

   //Sets sample rate to 8000/1+7 = 1000Hz
   master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_SMPLRT_DIV, 0x07);
   //Disable FSync, 256Hz DLPF, DLPF_CFG =4
   master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_CONFIG, 0x00);
   //Disable gyro self tests, scale of 500 degrees/s
```

```
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_GYRO_CONFIG,
0b00001000);
  //Disable accel self tests, scale of +-2g, no DHPF
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_ACCEL_CONFIG, 0x00);

  //Freefall threshold of |0mg|
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_FF_THR, 0x00);
  //Freefall duration limit of 0
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_FF_DUR, 0x00);
  //Motion threshold of 0mg
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_MOT_THR, 0x00);
  //Motion duration of 0s
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_MOT_DUR, 0x00);
  //Zero motion threshold
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_ZRMOT_THR, 0x00);
  //Zero motion duration threshold
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_ZRMOT_DUR, 0x00);
  //Disable sensor output to FIFO buffer
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_FIFO_EN, 0x00);

  //AUX I2C setup
  //Sets AUX I2C to single master control, plus other config
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_MST_CTRL, 0x00);
  //Setup AUX I2C slaves
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_ADDR, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_REG, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_CTRL, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_ADDR, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_REG, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_CTRL, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_ADDR, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_REG, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_CTRL, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_ADDR, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_REG, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_CTRL, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_ADDR, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_REG, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_DO, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_CTRL, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV4_DI, 0x00);

  //MPU6050_RA_I2C_MST_STATUS //Read-only
  //Setup INT pin and AUX I2C pass through
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_INT_PIN_CFG, 0x00);
  //Enable data ready interrupt
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_INT_ENABLE, 0x00);

  //Slave out, dont care
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV0_DO, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV1_DO, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV2_DO, 0x00);
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_SLV3_DO, 0x00);
  //More slave config
  master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_I2C_MST_DELAY_CTRL,
0x00);
```

```
    //Reset sensor signal paths
    master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_SIGNAL_PATH_RESET,
0x00);
    //Motion detection control
    master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_MOT_DETECT_CTRL, 0x00);
    //Disables FIFO, AUX I2C, FIFO and I2C reset bits to 0
    master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_USER_CTRL, 0x00);
    //Sets clock source to gyro reference w/ PLL
    master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_PWR_MGMT_1,
0b00000010);
    //Controls frequency of wakeups in accel low power mode plus the sensor standby modes
    master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_PWR_MGMT_2, 0x00);

    //Data transfer to and from the FIFO buffer
    master_I2C_write_byte(MPU6050_ADDRESS, MPU6050_RA_FIFO_R_W, 0x00);
    //MPU6050_RA_WHO_AM_I                   //Read-only, I2C address
}



void Calibrate_Gyros(void)
{
    int GYRO_XOUT_H,GYRO_XOUT_L,GYRO_YOUT_H, GYRO_YOUT_L, GYRO_ZOUT_H,
GYRO_ZOUT_L;
    int OFFSET_X, OFFSET_Y, OFFSET_Z;
    int x = 0;

    for(x = 0; x<1000; x++)
    {
    GYRO_XOUT_H = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_XOUT_H);
    GYRO_XOUT_L =  master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_XOUT_L);
    GYRO_YOUT_H = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_YOUT_H);
    GYRO_YOUT_L = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_YOUT_L);
    GYRO_ZOUT_H = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_ZOUT_H);
    GYRO_ZOUT_L = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_ZOUT_L);

        OFFSET_X += ((GYRO_XOUT_H<<8)|GYRO_XOUT_L);
    OFFSET_Y += ((GYRO_YOUT_H<<8)|GYRO_YOUT_L);
    OFFSET_Z += ((GYRO_ZOUT_H<<8)|GYRO_ZOUT_L);
    }
    OFFSET_X = OFFSET_X/1000;
    OFFSET_Y = OFFSET_Y/1000;
    OFFSET_Z = OFFSET_Z/1000;


        sensor_data_ptr->angular_velocity_X = OFFSET_X;
        sensor_data_ptr->angular_velocity_Y = OFFSET_Y;
        sensor_data_ptr->angular_velocity_Z = OFFSET_Z;
}
```

```
//Gets raw accelerometer data, performs no processing
int Get_Accel_Values(void)
{
   int ACCEL_XOUT_H,ACCEL_XOUT_L,ACCEL_YOUT_H,
ACCEL_YOUT_L,ACCEL_ZOUT_H,ACCEL_ZOUT_L;

   ACCEL_XOUT_H = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_XOUT_H);
   ACCEL_XOUT_L = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_XOUT_L);
   ACCEL_YOUT_H = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_YOUT_H);
   ACCEL_YOUT_L = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_YOUT_L);
   ACCEL_ZOUT_H = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_ZOUT_H);
   ACCEL_ZOUT_L = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_ACCEL_ZOUT_L);

   sensor_data_ptr->accel_X = ((ACCEL_XOUT_H<<8)|ACCEL_XOUT_L);
   sensor_data_ptr->accel_Y = ((ACCEL_YOUT_H<<8)|ACCEL_YOUT_L);
   sensor_data_ptr->accel_Z = ((ACCEL_ZOUT_H<<8)|ACCEL_ZOUT_L);

      return 1;
}
//Converts the already acquired accelerometer data into 3D euler angles
//void Get_Accel_Angles(void){
//  sensor_data_ptr->roll = 57.295*atan((float)sensor_data_ptr->accel_Y/
sqrt(pow((float)sensor_data_ptr->accel_Z,2)+pow((float)sensor_data_ptr->accel_X,2)));
//
//        sensor_data_ptr->pitch = 57.295*atan((float)-sensor_data_ptr->accel_X/
sqrt(pow((float)sensor_data_ptr->accel_Z,2)+pow((float)sensor_data_ptr->accel_Y,2)));
//
//}

//Function to read the gyroscope rate data and convert it into degrees/s

int Get_Gyro_Rates(void)
{
   int GYRO_XOUT_H, GYRO_XOUT_L, GYRO_YOUT_H, GYRO_YOUT_L, GYRO_ZOUT_H,
GYRO_ZOUT_L;
   int GYRO_XOUT, GYRO_YOUT, GYRO_ZOUT;

   GYRO_XOUT_H = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_XOUT_H);
   GYRO_XOUT_L = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_XOUT_L);
   GYRO_YOUT_H = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_YOUT_H);
   GYRO_YOUT_L = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_YOUT_L);
   GYRO_ZOUT_H = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_ZOUT_H);
   GYRO_ZOUT_L = master_I2C_read_byte(MPU6050_ADDRESS,
MPU6050_RA_GYRO_ZOUT_L);
```

```
    GYRO_XOUT = ((GYRO_XOUT_H<<8)|GYRO_XOUT_L) - GYRO_XOUT_OFFSET;
    GYRO_YOUT = ((GYRO_YOUT_H<<8)|GYRO_YOUT_L) - GYRO_YOUT_OFFSET;
    GYRO_ZOUT = ((GYRO_ZOUT_H<<8)|GYRO_ZOUT_L) - GYRO_ZOUT_OFFSET;


    //GYRO_XRATE = (float)GYRO_XOUT/gyro_xsensitivity;
    //GYRO_YRATE = (float)GYRO_YOUT/gyro_ysensitivity;
    //GYRO_ZRATE = (float)GYRO_ZOUT/gyro_zsensitivity;

        sensor_data_ptr->angular_velocity_X  =
(float)((GYRO_XOUT_H<<8)|GYRO_XOUT_L)/GYRO_SENSITIVITY;
        sensor_data_ptr->angular_velocity_Y  =
(float)((GYRO_YOUT_H<<8)|GYRO_YOUT_L)/GYRO_SENSITIVITY;
        sensor_data_ptr->angular_velocity_Z  =
(float)((GYRO_ZOUT_H<<8)|GYRO_ZOUT_L)/GYRO_SENSITIVITY;

        return 1;
}
```

MPU_6050.h

```
/mpu_6050 header file
#ifndef MPU_6050_H
#define MPU_6050_H

#include "routines.h"
#define MPU6050_ADDRESS 0x68 // Address with end write bit
#define MPU6050_RA_XG_OFFS_TC 0x00 //[7] PWR_MODE, [6:1] XG_OFFS_TC, [0]
OTP_BNK_VLD
#define MPU6050_RA_YG_OFFS_TC 0x01 //[7] PWR_MODE, [6:1] YG_OFFS_TC, [0]
OTP_BNK_VLD
#define MPU6050_RA_ZG_OFFS_TC 0x02 //[7] PWR_MODE, [6:1] ZG_OFFS_TC, [0]
OTP_BNK_VLD
#define MPU6050_RA_X_FINE_GAIN 0x03 //[7:0] X_FINE_GAIN
#define MPU6050_RA_Y_FINE_GAIN 0x04 //[7:0] Y_FINE_GAIN
#define MPU6050_RA_Z_FINE_GAIN 0x05 //[7:0] Z_FINE_GAIN
#define MPU6050_RA_XA_OFFS_H 0x06 //[15:0] XA_OFFS
#define MPU6050_RA_XA_OFFS_L_TC 0x07
#define MPU6050_RA_YA_OFFS_H 0x08 //[15:0] YA_OFFS
#define MPU6050_RA_YA_OFFS_L_TC 0x09
#define MPU6050_RA_ZA_OFFS_H 0x0A //[15:0] ZA_OFFS
#define MPU6050_RA_ZA_OFFS_L_TC 0x0B
#define MPU6050_RA_XG_OFFS_USRH 0x13 //[15:0] XG_OFFS_USR
#define MPU6050_RA_XG_OFFS_USRL 0x14
#define MPU6050_RA_YG_OFFS_USRH 0x15 //[15:0] YG_OFFS_USR
#define MPU6050_RA_YG_OFFS_USRL 0x16
#define MPU6050_RA_ZG_OFFS_USRH 0x17 //[15:0] ZG_OFFS_USR
#define MPU6050_RA_ZG_OFFS_USRL 0x18
#define MPU6050_RA_SMPLRT_DIV 0x19
#define MPU6050_RA_CONFIG 0x1A
#define MPU6050_RA_GYRO_CONFIG 0x1B
#define MPU6050_RA_ACCEL_CONFIG 0x1C
#define MPU6050_RA_FF_THR 0x1D
#define MPU6050_RA_FF_DUR 0x1E
#define MPU6050_RA_MOT_THR 0x1F
#define MPU6050_RA_MOT_DUR 0x20
```

```
#define MPU6050_RA_ZRMOT_THR 0x21
#define MPU6050_RA_ZRMOT_DUR 0x22
#define MPU6050_RA_FIFO_EN 0x23
#define MPU6050_RA_I2C_MST_CTRL 0x24
#define MPU6050_RA_I2C_SLV0_ADDR 0x25
#define MPU6050_RA_I2C_SLV0_REG 0x26
#define MPU6050_RA_I2C_SLV0_CTRL 0x27
#define MPU6050_RA_I2C_SLV1_ADDR 0x28
#define MPU6050_RA_I2C_SLV1_REG 0x29
#define MPU6050_RA_I2C_SLV1_CTRL 0x2A
#define MPU6050_RA_I2C_SLV2_ADDR 0x2B
#define MPU6050_RA_I2C_SLV2_REG 0x2C
#define MPU6050_RA_I2C_SLV2_CTRL 0x2D
#define MPU6050_RA_I2C_SLV3_ADDR 0x2E
#define MPU6050_RA_I2C_SLV3_REG 0x2F
#define MPU6050_RA_I2C_SLV3_CTRL 0x30
#define MPU6050_RA_I2C_SLV4_ADDR 0x31
#define MPU6050_RA_I2C_SLV4_REG 0x32
#define MPU6050_RA_I2C_SLV4_DO 0x33
#define MPU6050_RA_I2C_SLV4_CTRL 0x34
#define MPU6050_RA_I2C_SLV4_DI 0x35
#define MPU6050_RA_I2C_MST_STATUS 0x36
#define MPU6050_RA_INT_PIN_CFG 0x37
#define MPU6050_RA_INT_ENABLE 0x38
#define MPU6050_RA_DMP_INT_STATUS 0x39
#define MPU6050_RA_INT_STATUS 0x3A
#define MPU6050_RA_ACCEL_XOUT_H 0x3B
#define MPU6050_RA_ACCEL_XOUT_L 0x3C
#define MPU6050_RA_ACCEL_YOUT_H 0x3D
#define MPU6050_RA_ACCEL_YOUT_L 0x3E
#define MPU6050_RA_ACCEL_ZOUT_H 0x3F
#define MPU6050_RA_ACCEL_ZOUT_L 0x40
#define MPU6050_RA_TEMP_OUT_H 0x41
#define MPU6050_RA_TEMP_OUT_L 0x42
#define MPU6050_RA_GYRO_XOUT_H 0x43
#define MPU6050_RA_GYRO_XOUT_L 0x44
#define MPU6050_RA_GYRO_YOUT_H 0x45
#define MPU6050_RA_GYRO_YOUT_L 0x46
#define MPU6050_RA_GYRO_ZOUT_H 0x47
#define MPU6050_RA_GYRO_ZOUT_L 0x48
#define MPU6050_RA_EXT_SENS_DATA_00 0x49
#define MPU6050_RA_EXT_SENS_DATA_01 0x4A
#define MPU6050_RA_EXT_SENS_DATA_02 0x4B
#define MPU6050_RA_EXT_SENS_DATA_03 0x4C
#define MPU6050_RA_EXT_SENS_DATA_04 0x4D
#define MPU6050_RA_EXT_SENS_DATA_05 0x4E
#define MPU6050_RA_EXT_SENS_DATA_06 0x4F
#define MPU6050_RA_EXT_SENS_DATA_07 0x50
#define MPU6050_RA_EXT_SENS_DATA_08 0x51
#define MPU6050_RA_EXT_SENS_DATA_09 0x52
#define MPU6050_RA_EXT_SENS_DATA_10 0x53
#define MPU6050_RA_EXT_SENS_DATA_11 0x54
#define MPU6050_RA_EXT_SENS_DATA_12 0x55
#define MPU6050_RA_EXT_SENS_DATA_13 0x56
#define MPU6050_RA_EXT_SENS_DATA_14 0x57
#define MPU6050_RA_EXT_SENS_DATA_15 0x58
```

```
#define MPU6050_RA_EXT_SENS_DATA_16 0x59
#define MPU6050_RA_EXT_SENS_DATA_17 0x5A
#define MPU6050_RA_EXT_SENS_DATA_18 0x5B
#define MPU6050_RA_EXT_SENS_DATA_19 0x5C
#define MPU6050_RA_EXT_SENS_DATA_20 0x5D
#define MPU6050_RA_EXT_SENS_DATA_21 0x5E
#define MPU6050_RA_EXT_SENS_DATA_22 0x5F
#define MPU6050_RA_EXT_SENS_DATA_23 0x60
#define MPU6050_RA_MOT_DETECT_STATUS 0x61
#define MPU6050_RA_I2C_SLV0_DO 0x63
#define MPU6050_RA_I2C_SLV1_DO 0x64
#define MPU6050_RA_I2C_SLV2_DO 0x65
#define MPU6050_RA_I2C_SLV3_DO 0x66
#define MPU6050_RA_I2C_MST_DELAY_CTRL 0x67
#define MPU6050_RA_SIGNAL_PATH_RESET 0x68
#define MPU6050_RA_MOT_DETECT_CTRL 0x69
#define MPU6050_RA_USER_CTRL 0x6A
#define MPU6050_RA_PWR_MGMT_1 0x6B
#define MPU6050_RA_PWR_MGMT_2 0x6C
#define MPU6050_RA_BANK_SEL 0x6D
#define MPU6050_RA_MEM_START_ADDR 0x6E
#define MPU6050_RA_MEM_R_W 0x6F
#define MPU6050_RA_DMP_CFG_1 0x70
#define MPU6050_RA_DMP_CFG_2 0x71
#define MPU6050_RA_FIFO_COUNTH 0x72
#define MPU6050_RA_FIFO_COUNTL 0x73
#define MPU6050_RA_FIFO_R_W 0x74
#define MPU6050_RA_WHO_AM_I 0x75


void Setup_MPU6050(void);
void Calibrate_Gyros(void);
int Get_Accel_Values(void);
int Get_Gyro_Rates(void);
void Get_Accel_Angles(void);

#endif  /* MPU_6050_H */
```

Complementary_filter.c

```
#include "complementary_filter.h"
#include "i2c.h"
#include <math.h>

#define GYROSCOPE_SENSITIVITY 0.00875

#define M_PI 3.14159265

#define dt 0.07518796992 //13 HZ

#define alpha 0.15

#define round(x) ((x)>=0?(int)((x)+0.5):(int)((x)-0.5))

float fXg = 0;
```

```
float fYg = 0;
float fZg = 0;

void ComplementaryFilter()
{

   //using low pass-filter
   fXg  = sensor_data_ptr->accel_X * alpha + ( fXg * (1.0 - alpha));
   fYg  = sensor_data_ptr->accel_Y * alpha + ( fYg * (1.0 - alpha));
   fZg  = sensor_data_ptr->accel_Z * alpha + ( fZg * (1.0 - alpha));

   sensor_data_ptr->roll  = (atan2(-fXg, fZg))*180/M_PI;
   //sensor_data_ptr->pitch  = (atan2(fYg, fZg))*180/M_PI;
   sensor_data_ptr->pitch = (atan2(fYg, sqrt(fXg*fXg + fZg*fZg)))*180/M_PI;

   //heading in Z
    sensor_data_ptr->yaw = atan2(sensor_data_ptr->compass_X,sensor_data_ptr->compass_Y)
          * 180 / M_PI +180 ;

//sensor_data_ptr->roll = sensor_data_ptr->angular_velocity_X;
//sensor_data_ptr->pitch = sensor_data_ptr->angular_velocity_Y;
//sensor_data_ptr->yaw = sensor_data_ptr->angular_velocity_Z;


//   if(sensor_data_ptr->compass_Y < 0){
//         sensor_data_ptr->yaw = 270 - atan2(sensor_data_ptr->compass_X,sensor_data_ptr-
>compass_Y)
//         * 180 / M_PI ;
//   }
//   else if (sensor_data_ptr->compass_Y > 0){
//         sensor_data_ptr->yaw = 90 - atan2(sensor_data_ptr->compass_X,sensor_data_ptr-
>compass_Y)
//         * 180 / M_PI ;
//   }
//   else if ((sensor_data_ptr->compass_Y == 0) &&  (sensor_data_ptr->compass_X < 0)){
//         sensor_data_ptr->yaw = 180;
//   }
//   else if ((sensor_data_ptr->compass_Y == 0) && (sensor_data_ptr->compass_X > 0)){
//         sensor_data_ptr->yaw = 0;
//   }


   //Direction (y>0) = 90 - [arcTAN(x/y)]*180/Pi
   //Direction (y<0) = 270 - [arcTAN(x/y)]*180/Pi


    // Integrate the gyroscope data -> int(angularSpeed) = angle

    //sensor_data_ptr->pitch += ((float)sensor_data_ptr->angular_velocity_X  ) * dt; // Angle
around the X-axis
    //sensor_data_ptr->roll  -= ((float)sensor_data_ptr->angular_velocity_Y  ) * dt;  // Angle
around the Y-axis
    //sensor_data_ptr->yaw  += ((float)sensor_data_ptr->angular_velocity_Z ) * dt;


    //sensor_data_ptr->pitch = Round(sensor_data_ptr->pitch_g);
```

```
    //->roll =  Round(sensor_data_ptr->roll_g);
    // Turning around the X axis results in a vector on the Y-axis
  // sensor_data_ptr->pitch = atan2((float)sensor_data_ptr->accel_Y, (float)sensor_data_ptr->accel_Z) * 180 / M_PI;
    //  sensor_data_ptr->pitch_g = sensor_data_ptr->pitch_g * 0.98 + pitchAcc * 0.02;

    // Turning around the Y axis results in a vector on the X-axis

   // sensor_data_ptr->roll = atan2((float)sensor_data_ptr->accel_X, (float)sensor_data_ptr->accel_Z) * 180 / M_PI;

    //sensor_data_ptr->roll = sensor_data_ptr->roll_gy * 0.70 + sensor_data_ptr->roll_ac * 0.30;

}

int Round(float myfloat)
{
  return (int)(myfloat + 0.5);
}

// fusion final
//         Compute the final sensor outputs that feed into I2C to the flight controller
//
// PARAM   sensor_data_ptr: Pointer to the sensor data storage
//         sensor_result_ptr: Pointer to the sensor final result struct
void fusion_final(void)
{

    sensor_result_ptr->pitch =  (long)(sensor_data_ptr->pitch * 1000);
    sensor_result_ptr->roll = (long)(sensor_data_ptr->roll * 1000);
    sensor_result_ptr->yaw =  (long)(sensor_data_ptr->yaw * 1000);

}
```

Complementary_filter.h

```
#ifndef COMPLEMENTARY_FILTER_H
#define COMPLEMENTARY_FILTER_H
#include "routines.h"

void ComplementaryFilter(void);
int Round(float myfloat);
void fusion_final(void);

extern float fXg;
extern float fYg;
extern float fZg;

#endif  /* COMPLEMENTARY_FILTER_H */
```

uart.c

```
#include "uart.h"

// constraints:
```

```
 int count_sensor = 0;
int uart_task(void)
{
    double n;

    //n = ;
    //baud_test();
   // printf("\033[2J");

    if (error == NO_ERROR) {

//          if(count_sensor == 10)
//          {
            //printf("%d" ,sensor_data_ptr->angular_velocity_X);
             // printf("%d" ,sensor_data_ptr->angular_velocity_Y);
           // printf("%d" ,sensor_data_ptr->angular_velocity_Z);


//
//          printFloat(sensor_data_ptr->angular_velocity_X);
//          printFloat(sensor_data_ptr->angular_velocity_Y);
//          printFloat(sensor_data_ptr->angular_velocity_Z);

           //printFloat(sensor_data_ptr->pitch);
           //printFloat(sensor_data_ptr->roll);
            //printFloat(sensor_data_ptr->yaw);

        //printf("%d, %d, %d\n\r",sensor_data_ptr->compass_X, sensor_data_ptr->compass_Y,
sensor_data_ptr->compass_Z);

         //printFloat(sensor_data_ptr->compass_X);
         //printFloat(sensor_data_ptr->compass_Y);
         //printFloat(sensor_data_ptr->compass_Z);
          printf("%lu.\n\r",  sensor_result_ptr->yaw);
         printf("\n\r");
//          }
//          count_sensor++;
       //printFloat(sensor_data_ptr->compass_X);
       //printFloat(sensor_data_ptr->compass_Y);
       //printFloat(sensor_data_ptr->compass_Z);

    } else if (error == ERROR_I2C_READ_TIMEOUT) {
        printf("\n\r Error: Read I2C timeout.");
    }

    return 1;
}

//transmit 0x55 to test baud rate of TX
void baud_test(void)
{
    char test_byte = 't';
    _user_putc(test_byte);
}
```

```
//Low level protocols

//enable the receiver
void RX_open(void) {
    RCSTA1bits.CREN1 = 1;
    PIE1bits.RC1IE = 1;
    PIR1bits.RC1IF = 0;
    remaining_buffer_size = BUFFER_SIZE;
}

// close RX port when there are command to be executed
void RX_close(void) {
    RCSTA1bits.CREN1 = 0;
    PIE1bits.RC1IE = 0;
}

// pull the a byte out of the RX register and append it
// to the command string if there is no frame error
void RX_buffer_push(char data_byte) {
    *(RX_buffer_ptr + BUFFER_SIZE - remaining_buffer_size) = data_byte;
    remaining_buffer_size--;
    if (!remaining_buffer_size) {
        RX_close();
    }
}

void uart_init(void) {
    // Asynchronous mode, high speed, 9th bit-disabled
    TXSTA1 =   0b10100101;
    // 8-bit reception, clear framing and overrun error
    RCSTA1 =   0b10011000;
    // TX/RX high true. 1 as idle bit. 8-bit baud
    // rate generator, auto baud disabled
    BAUDCON1 = 0b01000000;
    SPBRGH1= 0x00;
    SPBRG1 = 0xCF;
        RX_open();
}

// send out a byte via uart or LCD, used as stdoutput
void _user_putc(char TX_byte)
{
    // turning on transmitting LED
    while(!TXSTAbits.TRMT) {
            if (sampling_flag) {
                timer_rst();
            }
        }
    TXREG1 = TX_byte;
}



void printFloat(double fInput)
{
    //The number is converted to two parts.
```

```
    long lWhole=(long)((double)fInput);
    long ulPart=(long)((double)fInput*100)-lWhole*100;

    printf(" %li.%li",lWhole,ulPart);

}
```

uart.h

```
#ifndef UART_H
#define UART_H

#define BUFFER_SIZE 1

// REPORT_FLAG indicate what the terminal
// is going to output

// The terminal sends the welcome msg
#define REPORT_FLAG_WELCOME 0
// The terminal is waiting for changes
#define REPORT_FLAG_WAIT        1
// REPORT_FLAG_REPORT means the terminal is scanning things
#define REPORT_FLAG_REPORT  2
#define REPORT_FLAG_IDLE        3
#define REPORT_FLAG_T   4
#define REPORT_FLAG_C   5
#define REPORT_FLAG_S   6
#define REPORT_FLAG_F   7
#define REPORT_FLAG_SCAN        8

#define BYTE_INSTRUCTION_REPORT 'r'
#define BYTE_INSTRUCTION_REPORT_ALL 'a'
#define BYTE_INSTRUCTION_SCAN 's'
#define BYTE_INSTRUCTION_TEMP_UNIT 't'
#define BYTE_INSTRUCTION_T '1'
#define BYTE_INSTRUCTION_C '2'
#define BYTE_INSTRUCTION_S '3'
#define BYTE_INSTRUCTION_F '4'
#define BYTE_INSTRUCTION_NEXT_UNIT 'k'
#define BYTE_INSTRUCTION_PREV_UNIT 'j'
#define BYTE_INSTRUCTION_NEXT_SIZE 'm'
#define BYTE_INSTRUCTION_PREV_SIZE 'n'

#define ACCURACY 2

#include "routines.h"

//global variables
extern char* RX_buffer_ptr;
extern unsigned int report_flag;
extern volatile int remaining_buffer_size;
extern volatile unsigned int RX_buffer_full;
extern volatile unsigned char byte_command;
extern unsigned int report_type;
```

```
// public functions
void uart_init(void);
void baud_test(void);
int uart_task(void);
void RX_ISR(void);

void printFloat(double fInput);


// private functions
void RX_buffer_push(char RX_buffer_push);
void _user_putc(char TX_byte);
void RX_open(void);
void RX_close(void);
unsigned int verify_data(char* data_ptr);



#endif
```

sensing_unit.c

```
// EE478 Lab2
// Measurement Unit

/* Compile options:  -ml (Large code model) */

// PIC18F25K22 Configuration Bit Settings
// CONFIG1H
//pragma config FOSC = ECHP
#pragma config FOSC = INTIO7   // Oscillator Selection bits (EC oscillator (high power, >16
MHz))
#pragma config PLLCFG = ON         // 4X PLL Enable (Oscillator used directly)
#pragma config PRICLKEN = ON      // Primary clock enable bit (Primary clock enabled)
#pragma config FCMEN = OFF         // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock
Monitor disabled)
#pragma config IESO = OFF          // Internal/External Oscillator Switch-over bit (Oscillator
Switch-over mode disabled)

// CONFIG2L
#pragma config PWRTEN = OFF
// Power-up Timer Enable bit (Power up timer disabled)
#pragma config BOREN = OFF  // Brown-out Reset Enable bits (Brown-out Reset enabled in
hardware only (SBOREN is disabled))
#pragma config BORV = 190          // Brown Out Reset Voltage bits (VBOR set to 1.90 V
nominal)

// CONFIG2H
#pragma config WDTEN = OFF         // Watchdog Timer Enable bits (Watch dog timer is always
disabled. SWDTEN has no effect.)
#pragma config WDTPS = 32768       // Watchdog Timer Post-scale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC1  // CCP2 MUX bit (CCP2 input/output is multiplexed with
RC1)
```

```
#pragma config PBADEN = OFF       // PORTB A/D Enable bit (PORTB<5:0> pins are configured as analog input channels on Reset)
#pragma config CCP3MX = PORTB5  // P3A/CCP3 Mux bit (P3A/CCP3 input/output is multiplexed with RB5)
#pragma config HFOFST = ON        // HFINTOSC Fast Start-up (HFINTOSC output and ready status are not delayed by the oscillator stable status)
#pragma config T3CMX = PORTC0   // Timer3 Clock input mux bit (T3CKI is on RC0)
#pragma config P2BMX = PORTB5   // ECCP2 B output mux bit (P2B is on RB5)
#pragma config MCLRE = EXTMCLR  // MCLR Pin Enable bit (MCLR pin enabled, RE3 input pin disabled)

// CONFIG4L
#pragma config STVREN = ON        // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = OFF          // Single-Supply ICSP Enable bit (Single-Supply ICSP enabled if MCLRE is also 1)
#pragma config XINST = OFF        // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode disabled (Legacy mode))

// CONFIG5L
#pragma config CP0 = OFF          // Code Protection Block 0 (Block 0 (000800-001FFFh) not code-protected)
#pragma config CP1 = OFF          // Code Protection Block 1 (Block 1 (002000-003FFFh) not code-protected)
#pragma config CP2 = OFF          // Code Protection Block 2 (Block 2 (004000-005FFFh) not code-protected)
#pragma config CP3 = OFF          // Code Protection Block 3 (Block 3 (006000-007FFFh) not code-protected)

// CONFIG5H
#pragma config CPB = OFF          // Boot Block Code Protection bit (Boot block (000000-0007FFh) not code-protected)
#pragma config CPD = OFF          // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF         // Write Protection Block 0 (Block 0 (000800-001FFFh) not write-protected)
#pragma config WRT1 = OFF         // Write Protection Block 1 (Block 1 (002000-003FFFh) not write-protected)
#pragma config WRT2 = OFF         // Write Protection Block 2 (Block 2 (004000-005FFFh) not write-protected)
#pragma config WRT3 = OFF         // Write Protection Block 3 (Block 3 (006000-007FFFh) not write-protected)

// CONFIG6H
#pragma config WRTC = OFF         // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not write-protected)
#pragma config WRTB = OFF         // Boot Block Write Protection bit (Boot Block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF         // Data EEPROM Write Protection bit (Data EEPROM not write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF        // Table Read Protection Block 0 (Block 0 (000800-001FFFh) not protected from table reads executed in other blocks)
```

```
#pragma config EBTR1 = OFF          // Table Read Protection Block 1 (Block 1 (002000-
003FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR2 = OFF          // Table Read Protection Block 2 (Block 2 (004000-
005FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR3 = OFF          // Table Read Protection Block 3 (Block 3 (006000-
007FFFh) not protected from table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF          // Boot Block Table Read Protection bit (Boot Block
(000000-0007FFh) not protected from table reads executed in other blocks)

// ISR to reset other parts
#include "routines.h"

// allocate memory for sensor data
sensor_data_struct sensor_data;
sensor_data_struct* sensor_data_ptr;
sensor_result_struct sensor_result;
sensor_result_struct* sensor_result_ptr;

// global variables for scheduler
unsigned int curr_channel;
unsigned int system_status;
unsigned int sampling_flag;
unsigned int error;

// global variables for UART
volatile char RX_buffer[50];
volatile int remaining_buffer_size;
volatile unsigned int RX_buffer_full;
volatile unsigned char byte_command;
unsigned int report_type;
unsigned int target_ID;
char* RX_buffer_ptr;

// global variables misc sensors
unsigned int freqCounter;


// global variables for I2C
char I2C_TX_buffer[8];
char* I2C_TX_buffer_ptr;
char I2C_RX_buffer[40];
char* I2C_RX_buffer_ptr;

unsigned int report_flag;

void system_init(void);

void main(void)
{
   //count = 0;
    //system init
   system_init();
   // redirect output stream to _H_USER
   // initialize peripherals
```

```
    // initialize global variables
    done = 0;
    curr_channel = 0;
    RX_buffer_ptr = &(RX_buffer[0]);

    report_flag = 0;
    sampling_flag = 0;
    byte_command = '0';

    sensor_data_ptr->accel_X = 0;
    sensor_data_ptr->accel_Y = 0;
    sensor_data_ptr->accel_Z = 0;

    sensor_data_ptr->yaw = 0;
    sensor_data_ptr->pitch = 0;
    sensor_data_ptr->roll = 0;


    I2C_TX_buffer_ptr = &(I2C_TX_buffer[0]);
    I2C_RX_buffer_ptr = &(I2C_RX_buffer[0]);

    sensor_data_ptr = &sensor_data;
    sensor_data_ptr->angular_velocity_X = 0;
    sensor_data_ptr->angular_velocity_Y = 0;
    sensor_data_ptr->angular_velocity_Z = 0;

    sensor_result_ptr = &sensor_result;
    sensor_result_ptr->altitude = 0;
    sensor_result_ptr->pitch = 0;
    sensor_result_ptr->roll =0;
    sensor_result_ptr->yaw =0;

    error = NO_ERROR;

    // start measurements
    start_scheduler();
}

// power up initialization
void system_init(void)
{
    // system clk set-up
    OSCCON  = 0b11111000;
    OSCCON2 =          0b00000000;
    //low-frequency internal osc sourced from LFINTOSC
    OSCTUNEbits.INTSRC = 0;
    //disable PLL
    OSCTUNEbits.PLLEN = 1;
    // IO ports initialization
    // see document for exact assignments
    TRISA =     0b11000111;
    TRISB =     0xFF;
    TRISC =     0b11011000;
    ANSELA =    0b00000000;
```

```
   ANSELB =   0x00;
   ANSELC =   0x00;
   stdout = _H_USER;
   WPUB   =    0x00;
   IOCB   =    0x00;
   PORTAbits.RA5 = 1;


    //I2C for sensor card to flight control pic
    TRISBbits.RB2 = 1;
    TRISBbits.RB1 = 1;
    ANSELBbits.ANSB1 = 0; // digital output
    ANSELBbits.ANSB2 = 0;

   //uart_init();
   timer_init();
   I2C_slave_init();
   I2C_master_init();
   interrupt_init();
   sensors_init();



}

void bus_reset(void)
{
    TRISB = 0x00;
    PORTB = 0x55;
}

unsigned int bytes2int(unsigned char upper, unsigned char lower)
{
    unsigned int result;
    result = upper;
    result = result << 8 | lower;
    return result;
}


void sensors_init(void)
{
    //bmp085_init(&bmp);
    //gyro_init();
    //acceler_init();
    Setup_MPU6050();
    magneto_init();
}
```

scheduler.c

```
#include "scheduler.h"

unsigned int done;

void timer_init(void) {
```

```
    // Timer0, used to time motor 1
    T0CONbits.T08BIT = 1; // set timer0 in 8-bit mode
    T0CONbits.T0CS = 0; // set timer0 clk source as instruction cycle
    T0CONbits.PSA = 0; // enable timer0 pre-scale
    T0CONbits.T0SE = 0; // increment on low to high transition
    T0CONbits.T0PS = 0b001; // choose a pre-scale of 4
    // turn on Timer0
    T0CONbits.TMR0ON = 1;

    // Timer2, used as the scheduler timer

    // choose a prescale of 4
    T2CONbits.T2CKPS = 0b11;
    // turn on Timer2
    T2CONbits.TMR2ON = 1;
}

void start_scheduler(void) {
    while (1) {
        //timer interrupt
        if (sampling_flag == 1) {
            timer_rst();
            if (curr_channel == 1) {
                // used to indicate scheduler cycle in GPIO pin RA5
                update_status(SYSTEM_START);
                PORTAbits.RA5 = !PORTAbits.RA5;
            } else if (curr_channel == 100) {
                update_status(SYSTEM_GYRO);

            } else if (curr_channel == 200) {
                update_status(SYSTEM_ACCELERO);
                done = Get_Accel_Values();

            } else if (curr_channel == 300) {
               update_status(SYSTEM_COMPASS);
               done = get_compass();
              //done = Compass_test_single();
            } else if (curr_channel == 450) {
                update_status(SYSTEM_FUSION);
                //Calibrate_Gyros();
                ComplementaryFilter();
                fusion_final();
            } else if (curr_channel == 800) {
                update_status(SYSTEM_UART);
               // done = uart_task();
            }
        } else {
             //update status if a task terminate
            if (done && (check_status() == SYSTEM_UART) && TXSTAbits.TRMT) {
                update_status(SYSTEM_IDLE);
                done = 0;
            } else if (done && ((check_status() == SYSTEM_GYRO) ||
                    check_status() == SYSTEM_ACCELERO ||
                    check_status() == SYSTEM_COMPASS  ||
                    check_status() == SYSTEM_FUSION
                    )) {
```

```
            update_status(SYSTEM_IDLE);
            done = 0;
        }
    }
  }
}
// update system status to R5 RC2 RC1 RC0

void update_status(int status) {
  int status_out = 0;
  status_out = status_out | (status & 0x01) | (status & 0x02) | status & 0x04 | ((status & 0x08)
<< 2);
  PORTC = status_out;
}

int check_status(void) {
  return PORTCbits.RC0
      | (PORTCbits.RC1 << 1)
      | (PORTCbits.RC2 << 2)
      | (PORTCbits.RC5 << 5);
}

void timer_rst(void) {
  sampling_flag = 0;
  TMR0H = 0x00;
  TMR0L = 180;
  curr_channel = (curr_channel >= 1000) ? 0: curr_channel+1;
  // take data from analog channels
}

// delay n clk_cycle, update curr_chan;

void timer0_delay(int clk_cycle) {
  int count = 0;
  while (count < clk_cycle) {
    if (sampling_flag) {
      timer_rst();
      count++;
    }
  }
}

// delay operations until the master is ready.
void init_delay(void)
{
  timer0_delay(100000);
}
```

scheduler.h

```
// schedule the sensor module to achieve a certain sampling rate and low power consumption
#ifndef RESET_H
#define RESET_H

#include "routines.h"
```

```
void timer_init(void);
void start_scheduler(void);
void update_status(int status);
int check_status(void);
void timer_rst(void);

extern unsigned int done;
extern unsigned int curr_channel;
extern unsigned int freqCounter;
void init_delay(void);
#endif
```

routines.h

```
#ifndef ROUTINES_H
#define ROUTINES_H

//#include "adc.h"
#include "uart.h"
#include "scheduler.h"
#include "interrupts.h"
#include "intra_comm.h"
#include "MPU_6050.h"
#include "complementary_filter.h"
#include "sensor_magnetometer.h"

#include <stdio.h>
#include <p18F25K22.h>

extern unsigned int TX_buffer_size;
extern unsigned int RX_buffer_size;

// definitions
#define TIMER0_COUNTTO 125

#define SYSTEM_STATUS_MASK 0x27

// system status
#define SYSTEM_IDLE         0x00
#define SYSTEM_ACCELERO         0x01
#define SYSTEM_GYRO       0x02
#define SYSTEM_COMPASS          0x06
#define SYSTEM_START      0x07
#define SYSTEM_FUSION     0x05
#define SYSTEM_MASTER_READ 0x04
#define SYSTEM_UART       0x03
// error
#define NO_ERROR            0
#define ERROR_I2C_READ_TIMEOUT 1


// define sensor data storage block
typedef struct
{
    float angular_velocity_X;
    float angular_velocity_Y;
```

```
    float angular_velocity_Z;
    float compass_X;
    float compass_Y;
    float compass_Z;
    float accel_X;
    float accel_Y;
    float accel_Z;
    float roll;
    float pitch;
    //float roll_gy;
    //float pitch_gy;
    float roll_ac;
    float pitch_ac;
    float yaw;

    char data_size;
    unsigned int secret_code;

} sensor_data_struct;

// final result for the sensor processing unit
typedef struct
{
    long pitch;
    long roll;
    long yaw;
    long altitude;
} sensor_result_struct;

extern char* RX_buffer_ptr;
extern sensor_data_struct* sensor_data_ptr;
extern sensor_result_struct* sensor_result_ptr;
extern unsigned int report_flag;
extern volatile int remaining_buffer_size;
extern volatile unsigned int RX_buffer_full;
extern volatile unsigned char byte_command;
extern unsigned int report_type;
extern unsigned int error;

extern unsigned int sampling_flag;

extern char* I2C_TX_buffer_ptr;
extern char* I2C_RX_buffer_ptr;

void check_ack(void);
void sensors_init(void);

// public functions
void uart_init(void);
int uart_task(void);
void baud_test(void);

void interrupt_init(void);
void global_interrupt(void);
void myISR(void);
```

```
void timer_init(void);
void start_scheduler(void);
void timer0_delay(int clk_cycle);

void I2C_slave_init(void);
void I2C_master_init(void);

unsigned int bytes2int(unsigned char upper,unsigned char lower);
#endif
```

Control

Command line Graphical Interface for the quadcopter

Quad_GUI.py

```
##############################################################################
#                                            #
#                  Quad version 1.0                  #
#                     Python 2.7                  #
#                                            #
#                 EE 478 Autumn 2013                  #
#                                            #
##############################################################################

#!usr/bin/python2.7

import subprocess
import optparse
import re
```

```
import os
import argparse
import serial
import time
import RPi.GPIO as GPIO
import datetime
import logging
import threading
import curses
import Queue
import traceback

camera_task = 0
exit_threads = False

#################################
#                               #
#      call bash command        #
 #                               #
#################################
# compose a command list into a command and
# initiate a subprocess call
def compose(cl):
   cmd = " ".join(cl)
   print cmd
   subprocess.call(cmd, shell = True)


#################################
#                               #
#           Self_test           #
#     Test serial & GPIO        #
#                               #
#################################
def test():
   print 'PI test initiating...'
   print 'sending test serial to client'
   print 'blinking on GPIO21'
   blink(15)

# serial and GPIO testing
def blink(pin_num):
   while (1):
      GPIO.output(pin_num, True)
      time.sleep(1)
      GPIO.output(pin_num, False)
      time.sleep(1)
      ser.write('uuu')


#################################
```

```
#                       #
#         record        #
#    multimedia command     #
#                       #
#################################
def record():
  print 'start recording'
  timestamp = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d_%H-%M-2S')
  output_vid_path = '/home/pi/quadcopter/recordings/' + timestamp + '.h264'
  compose(['raspivid', '-o', output_vid_path])

#################################
#                       #
#      take_picture         #
#     photographic command   #
#                       #
#################################
def take_pic():
  print 'Cheeeeeeeeeeeeeeeeeese'
  timestamp = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d_%H-%M-
%S')
  output_image_path = '/home/pi/quadcopter/DCIM/pic' + timestamp + '.png'
  compose(['raspistill', '-o', output_image_path])

#################################
#                       #
#    clean video buffer     #
#                       #
#################################
def clean():
  print 'cleaning home/pi/quadcopter/recordings'
  compose('rm', '-i', 'home/pi/quadcopter/recordings/*')


#################################
#                       #
#  The serial read thread    #
#                       #
#################################

class SerialPortThread(threading.Thread):
  def __init__(self, port, logger, queue):
    threading.Thread.__init__(self)
    self._port = port
    self._logger = logger
    self._queue = queue

  def run(self):
    global exit_threads
    self._logger.info("SerialPortThread. Run.")
    while (not exit_threads):
```

```
        try:
          c = self._port.read()
          if (len(c) > 0):
            self._queue.put(c)
            self._logger.info("SerialPortThread. Editing. ")
        except:
          self._logger.info("SerialPortThread. In trouble. ")
      self._logger.info("SerialPortThread. Exiting")

#################################
#                               #
#   The keyboard thread         #
#                               #
#################################
class KeyboardThread(threading.Thread):
  def __init__(self, ui, logger, port, display):
    threading.Thread.__init__(self)
    self._ui = ui
    self._logger = logger
    self._port   = port
    self._display = display
    self._display.addstr(1,1, "Input:")

  def run(self):
    global exit_threads
    self._logger.info("KeyboardThread. Run.")
    while (not exit_threads):
      try:
        key = self._ui.getkey()
        self._logger.info("KeyboardThread. Event: >>" + key + "<<")
        if (key[0] == 'p'):
          take_pic()
          compose(['echo', 'hello'])
        elif (key[0] == 'v'):
          record()
        elif (key[0] == 'u'):
          clean()
        self._display.addstr(1,10,key[0])
        self._port.write(key[0])
        self._display.refresh()
        if ord(key) ==  27: #exit when esc was pressed
          self._logger.info("KeyboardThread. Exiting")
          exit_threads = True
      except:
        self._logger.info("KeyboardThread. Exception")
        exit_threads = True
        traceback.print_exc()


#################################
#                               #
```

```
#   The console thread       #
#                            #
##################################
class ConsoleThread(threading.Thread):
  line = 2

  def __init__(self, screen, logger, queue, program):
    threading.Thread.__init__(self)
    self._screen = screen
    self._logger = logger
    self._queue = queue
    self._program = program
    self._screen.addstr(0,1, 'Flight controller:')

  def run(self):
    global exit_threads
    self._logger.info("ConsoleThread.Run")
    y, x = self._screen.getmaxyx()
    pos = 1
    while (not exit_threads):
      try:
        e = self._queue.get(timeout = 0.1)
        self._logger.info("ConsoleThread: Input:>>" + str(e)+ "<<")
        if (self.line > y - 2):
          self.line = 1
          self._screen.clear()
          self._screen.addstr(0,1, 'Flight controller:')
          self._screen.refresh()
        if e == '\n':
          pos = 1
          self.line = self.line + 1
        elif (pos >= x-8):
          pos = 1
          self.line = self.line + 1
        else:
          self._logger.info("Char: " + str(ord(e)))
          self._screen.addstr(self.line, pos, e)
          self._screen.refresh()
          pos = pos+1
      except:
        pass

    self._logger.info("ConsoleThread. Exiting")



if __name__ == '__main__':
  #unload arguments
  # create argument parser and add argument lists
  par = argparse.ArgumentParser()
```

```
  par.add_argument("-m", "--mount_point", default = "/dev/ttyAMA0", help="choose the
mounting point of the serial port from /dev")
  par.add_argument("-b", "--baud_rate", default = "19200", help = "set the baud rate for serial
communication")
  par.add_argument("-t", "--serial_timeout", default = "60", help = "set the timeout for seiral
communication")
  par.add_argument("-l", "--video_duration", default = "100", help = "set the length of the video
being recorded")
  par.add_argument("-c", "--connection_test", action="store_true", help = "add the option to let
the quadcopter test its GPIO and serial IO")

  # unloading arguments
  args = par.parse_args()
  serial_port = args.mount_point
  baud_rate = int(args.baud_rate)
  timeout = int(args.serial_timeout)

  # initiate GPIO pins
  GPIO.setwarnings(False) #disable the warnings
  GPIO.setmode(GPIO.BOARD)
  GPIO.setup(13, GPIO.OUT)
  GPIO.setup(15, GPIO.OUT)
  GPIO.setup(16, GPIO.OUT)
  GPIO.setup(18, GPIO.OUT)

  # init the queue
  queue = Queue.Queue()

  # initiate serial communication
  ser = serial.Serial(serial_port, baud_rate, timeout = timeout)

  # init the logger
  # delete the previous log file
  print 'starting quad...'
  compose(['rm', '/home/pi/quad.log'])

  # init the new log file
  logging.basicConfig(format='%(asctime)-6s: %(name)s - %(levelname)s -%(message)s',
level=5, filename="quad.log")

  if (args.connection_test):
    test()
  else:

    # init curses
    quad_GUI = curses.initscr()
    # turn off keypress echoing
    curses.noecho()
    # set cursor visibility to 0(invisible), 1(normal), 2(very visible)
    curses.curs_set(0)
    # turn off line buffer
```

```
    try:
      curses.raw()
      quad_GUI.keypad(0)
      quad_GUI.clear()
      y, x = quad_GUI.getmaxyx()

      #creat the input_box
      input_box= quad_GUI.subwin(3, x-5, y/6*5, 2)
      input_box.box()

      #creat the console window in the middle
      console = quad_GUI.subwin(14, x-5, y/5, 2)

      c_y, c_x = console.getmaxyx()

      #decorations
      quad_GUI.hline(2,1, curses.ACS_HLINE, 77)
      quad_GUI.box()

      quad_GUI.addstr(1,1, 'Quad v1.0   EE478 UW    Author Phongsagon.S, Rebecca.C,
Jian.M')
      console.refresh()
      input_box.refresh()
      quad_GUI.refresh()

      threads = []
      thread1 = SerialPortThread(ser, logging, queue)
      threads.append(thread1)

      thread2 = KeyboardThread(console, logging, ser, input_box)
      threads.append(thread2)

      thread3 = ConsoleThread(console, logging, queue, quad_GUI)
      threads.append(thread3)

      for t in threads:
        t.setDaemon(False)
        t.start()

      GPIO.output(15, True)
      # spend the boring time drawing animations while waiting...

      for i,t in enumerate(threads) :
        t.join()

      curses.echo()
      curses.nocbreak()
      curses.endwin()         #terminate curses
      GPIO.output(15, False)

    except:
```

```
        # in the event of error, restore the terminal to sane state
        curses.echo()
        curses.nocbreak()
        curses.endwin()
        traceback.print_exc()    #print the exception
        print 'quad_GUI crashed'
```

Flight Controller firmware

*flight_contrller.c*

```
// EE478 Lab2
// Measurement Unit

/* Compile options:  -ml (Large code model) */

// PIC18F25K22 Configuration Bit Settings
// CONFIG1H
//pragma config FOSC = ECHP
#pragma config FOSC = INTIO7   // Oscillator Selection bits (EC oscillator (high power, >16
MHz))
#pragma config PLLCFG = ON    // 4X PLL Enable (Oscillator used directly)
#pragma config PRICLKEN = ON    // Primary clock enable bit (Primary clock enabled)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor
disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switch-over bit (Oscillator Switch-
over mode disabled)

// CONFIG2L
#pragma config PWRTEN = OFF
// Power-up Timer Enable bit (Power up timer disabled)
#pragma config BOREN = OFF  // Brown-out Reset Enable bits (Brown-out Reset enabled in
hardware only (SBOREN is disabled))
#pragma config BORV = 190       // Brown Out Reset Voltage bits (VBOR set to 1.90 V nominal)

// CONFIG2H
#pragma config WDTEN = OFF      // Watchdog Timer Enable bits (Watch dog timer is always
disabled. SWDTEN has no effect.)
#pragma config WDTPS = 32768    // Watchdog Timer Post-scale Select bits (1:32768)

// CONFIG3H
#pragma config CCP2MX = PORTC1  // CCP2 MUX bit (CCP2 input/output is multiplexed with
RC1)
#pragma config PBADEN = OFF      // PORTB A/D Enable bit (PORTB<5:0> pins are configured
as analog input channels on Reset)
#pragma config CCP3MX = PORTB5  // P3A/CCP3 Mux bit (P3A/CCP3 input/output is
multiplexed with RB5)
#pragma config HFOFST = ON      // HFINTOSC Fast Start-up (HFINTOSC output and ready
status are not delayed by the oscillator stable status)
#pragma config T3CMX = PORTC0   // Timer3 Clock input mux bit (T3CKI is on RC0)
#pragma config P2BMX = PORTB5   // ECCP2 B output mux bit (P2B is on RB5)
```

```
#pragma config MCLRE = EXTMCLR  // MCLR Pin Enable bit (MCLR pin enabled, RE3 input pin disabled)

// CONFIG4L
#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)
#pragma config LVP = OFF        // Single-Supply ICSP Enable bit (Single-Supply ICSP enabled if MCLRE is also 1)
#pragma config XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode disabled (Legacy mode))

// CONFIG5L
#pragma config CP0 = OFF        // Code Protection Block 0 (Block 0 (000800-001FFFh) not code-protected)
#pragma config CP1 = OFF        // Code Protection Block 1 (Block 1 (002000-003FFFh) not code-protected)
#pragma config CP2 = OFF        // Code Protection Block 2 (Block 2 (004000-005FFFh) not code-protected)
#pragma config CP3 = OFF        // Code Protection Block 3 (Block 3 (006000-007FFFh) not code-protected)

// CONFIG5H
#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot block (000000-0007FFh) not code-protected)
#pragma config CPD = OFF        // Data EEPROM Code Protection bit (Data EEPROM not code-protected)

// CONFIG6L
#pragma config WRT0 = OFF       // Write Protection Block 0 (Block 0 (000800-001FFFh) not write-protected)
#pragma config WRT1 = OFF       // Write Protection Block 1 (Block 1 (002000-003FFFh) not write-protected)
#pragma config WRT2 = OFF       // Write Protection Block 2 (Block 2 (004000-005FFFh) not write-protected)
#pragma config WRT3 = OFF       // Write Protection Block 3 (Block 3 (006000-007FFFh) not write-protected)

// CONFIG6H
#pragma config WRTC = OFF       // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) not write-protected)
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot Block (000000-0007FFh) not write-protected)
#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM not write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF      // Table Read Protection Block 0 (Block 0 (000800-001FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF      // Table Read Protection Block 1 (Block 1 (002000-003FFFh) not protected from table reads executed in other blocks)
```

```
#pragma config EBTR2 = OFF      // Table Read Protection Block 2 (Block 2 (004000-005FFFh) not protected from table reads executed in other blocks)
#pragma config EBTR3 = OFF      // Table Read Protection Block 3 (Block 3 (006000-007FFFh) not protected from table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF      // Boot Block Table Read Protection bit (Boot Block (000000-0007FFh) not protected from table reads executed in other blocks)

// ISR to reset other parts
#include "routines.h"
#include "controller.h"

// global variables for scheduler
unsigned int curr_channel;
unsigned int system_status;
unsigned int sampling_flag;
unsigned int error;

// global variables for UART
volatile char RX_buffer[10];
volatile int remaining_buffer_size;
volatile unsigned int RX_buffer_full;
volatile unsigned char byte_command;
unsigned int report_type;
char* RX_buffer_ptr;

// global storages for the controller/stabilizer
controller_parameters controller_param_struct;
quadcopter_intrinsics quadcopter_intrinsics_struct;
kinetic_errors kinetic_errors_struct;
controller_data controller_data_struct;
PID_storage roll_PID_struct;
PID_storage pitch_PID_struct;
PID_storage yaw_PID_struct;


// global variables for I2C
char I2C_TX_buffer[8];
char I2C_RX_buffer[40];
unsigned int test_motor_speed[4];

unsigned int report_flag;

void system_init(void);

void main(void)
{
  // system init
  system_init();
```

```
    // initialize global variables
    RX_buffer_ptr = &(RX_buffer[0]);
    report_flag = 0;
    sampling_flag = 0;

    // start measurements
    start_scheduler();
}

// power up initialization
void system_init(void)
{
    // system clk set-up
    OSCCON  =    0b11111000;
    OSCCON2 =    0b00000000;
    //low-frequency internal osc sourced from LFINTOSC
    OSCTUNEbits.INTSRC = 0;
    //enable PLL
    OSCTUNEbits.PLLEN = 1;
    // IO ports initialization
    // see document for exact assignments
    TRISA =     0b11000111;
    TRISB =     0x00;       //output
    TRISC =     0b11011000;
    ANSELA =    0b00000000;
    ANSELB =    0x00;
    ANSELC =    0x00;
    // redirect output stream to _H_USER
    stdout = _H_USER;
    WPUB   =    0x00;
    IOCB   =    0x00;
    PORTAbits.RA5 = 1;
    error = NO_ERROR;

    // init system modules
    stabilize_init(&kinetic_errors_struct, &(roll_PID_struct), &(pitch_PID_struct),
&(yaw_PID_struct));
    uart_init();
    timer_init();
    controller_init(&(quadcopter_intrinsics_struct), &(controller_data_struct),
&(controller_param_struct), &(test_motor_speed[0]));
    motors_init();
    interrupt_init();
    I2C_master_init(&(I2C_TX_buffer[0]), &(I2C_RX_buffer[0]));
}

void bus_reset(void)
{
     TRISB = 0x00;
     PORTB = 0x55;
}
```

```
unsigned int bytes2int(unsigned char upper, unsigned char lower)
{
    unsigned int result;
    result = upper;
    result = result << 8 | lower;
    return result;
}
```

*routines.h*

```
 #ifndef ROUTINES_H
#define ROUTINES_H

#include "uart.h"
#include "scheduler.h"
#include "interrupts.h"
#include "intra_comm.h"
#include "motor_drive.h"
#include "stabilizer.h"
#include "controller.h"
#include "PID.h"
#include <stdio.h>
#include <p18F25K22.h>

extern unsigned int TX_buffer_size;
extern unsigned int RX_buffer_size;

// definitions
#define TIMER0_COUNTTO 125

#define SYSTEM_STATUS_MASK 0x27

// system status
#define SYSTEM_IDLE        0x00
#define SYSTEM_STABILIZER  0x01
#define SYSTEM_CONTROLLER  0x02
#define SYSTEM_UART        0x03
#define SYSTEM_I2C_R       0x04
#define SYSTEM_MOTOR       0x05

// error
#define NO_ERROR                   0
#define SERIAL_TRANSMISSION_ERROR   1
#define I2C_TRANSMISSION_ERROR      2
#define INVALID_COMMAND             3
#define CANNOT_TERMINATE_SEQUENCE   4
#define HOLY_SHIT_ITS_GONNA_GO_DOWN 5

extern char* RX_buffer_ptr;
extern unsigned int report_flag;
```

```
extern volatile int remaining_buffer_size;
extern volatile unsigned int RX_buffer_full;
extern volatile unsigned char byte_command;
extern unsigned int report_type;
extern unsigned int error;

extern unsigned int sampling_flag;

void check_ack(void);

// public functions
void uart_init(void);
int uart_task(void);
void baud_test(void);

void interrupt_init(void);
void global_interrupt(void);
void myISR(void);

void timer_init(void);
void start_scheduler(void);
void timer0_delay(int clk_cycle);

void I2C_slave_init(void);
void I2C_master_init(char* TX_buffer, char* RX_buffer);

unsigned int bytes2int(unsigned char upper,unsigned char lower);
#endif
```

*controller.h*

```
#ifndef CONTROLLER_H
#define CONTROLLER_H

#include "routines.h"

#define MANUAL_CTRL 0
#define AUTOPILOT  1

// flight controller state
#define FCS_PWRON    0
#define FCS_ZERO     1
#define FCS_TAKEOFF  2
#define FCS_HOVERING 3
#define FCS_CRASH    4    //emergency landing
#define FCS_OFFSET   5
#define FCS_LANDING  6    //land gracefully
#define FCS_AUTO_F   7    //auto forward
#define FCS_AUTO_B   8    //auto backward
#define FCS_AUTO_L   9    //auto left
#define FCS_AUTO_R   10   //auto right
#define FCS_AUTO_U   11   //auto up
```

```
#define FCS_AUTO_D   12   //auto down
#define FCS_AUTO_YL  13   //auto yaw left
#define FCS_AUTO_YR  14   //auto yaw right
#define FCS_MOTOR_TEST   15

#define YAW_LOW_BOND -180   // -180 degree < Yaw < 180 degree
#define YAW_HIGH_BOND 180
#define PITCH_LOW_BOND -10  // -10 degree < Pitch < 10 degree
#define PITCH_HIGH_BOND 10
#define ROLL_LOW_BOND -10   //  -10 degree < Roll < 10 degree
#define ROLL_HIGH_BOND 10
#define MAX_LIFT 99999999 // No max lift limit
#define YAW_UNIT_CHANGE 30 // rotate the yaw by 30 degrees every time you press it.
#define ROLL_UNIT_CHANGE 1 // rotate the roll by 1 degrees every time you press it.
#define PITCH_UNIT_CHANGE 1 // rotate the pitch by 1 degrees every time you press it.
#define LIFT_UNIT_CHANGE 100 // increase 1 N everytime we press it.
#define MOTOR_TEST_STEP 50

typedef struct
{
   unsigned char  FCS; // Flight Controller State
   unsigned int  controller_counter; // used to time maneuvers
   double  roll_target; // -10 degrees to 10 degrees
   double  pitch_target; // -10 degrees to 10 degrees
   double  yaw_target;   // -179 degrees to 180 degrees
   double  lift; // 0 to MAX lift
   unsigned int  auto_manual_sel; // selecting autopilot or manual control
   unsigned char manual_is_pressed;
   unsigned char manual_key_pressed;
} controller_parameters;

typedef union
{
   struct {
      double U1;
      double U2;
      double U3;
      double U4;
   };
   struct {
      double roll;
      double pitch;
      double yaw;
      double altitude;
   };
   struct {
      double omega1square;
      double omega2square;
      double omega3square;
      double omega4square;
   };
```

```
} controller_data;

typedef struct
{
   double  m;  //mass of the quad
   double IXX; //moment of inertia of the quad around the roll axis
   double IYY; //moment of inertia of the quad around the pitch axis
   double IZZ; //moment of inertia of the quad around the yaw axis
   double d;   // drag factor
   double b;   // trust factor
   double l;   // distance between the center of the rotor to the center of the quadcopter fram
   double hl;  // hovering lift
} quadcopter_intrinsics;

extern controller_parameters* controller_param;
extern controller_data* ctrl_data;
extern quadcopter_intrinsics* quad_intrinsics;
extern unsigned int* test_speed;

unsigned int flight_control_routine(void);
void process_command(unsigned char byte_instruction);
void controller_init(quadcopter_intrinsics*, controller_data*, controller_parameters*, unsigned
int*);
void update_target_roll(double new_roll);
void update_target_pitch(double new_pitch);
void update_target_yaw(double new_yaw);
void update_lift(double new_lift);


// autopilot
void zero_quadcopter(void);
void take_off(void);
void crash_landing(void);
void controlled_landing(void);
void panorama_maneuver(void);
void foward(unsigned int distance);
void backward(unsigned int distance);
void left(unsigned int distance);
void right(unsigned int distance);
void up(unsigned int distance);
void down(unsigned int distance);
void yaw_CW(unsigned int degrees);
void yaw_CCW(unsigned int degrees);

// manuel control
void offset_clear(void);
void motor_test(unsigned int sign, unsigned int value, unsigned int motor_num);
void lift_offset(unsigned int sign, unsigned int value);
void roll_offset(unsigned int sign, unsigned int value);
void pitch_offset(unsigned int sign, unsigned int value);
void yaw_offset(unsigned int sign, unsigned int value);
```

```
void inverted_motion_matrix(void);


#endif
```

*controller.c*

```
#include "controller.h"

/**************************************************
*           EE478 quadcopter controller           *
* The motor speeds parameters are offsetted here    *
**************************************************/

controller_parameters* controller_param;
quadcopter_intrinsics* quad_intrinsics;
controller_data* ctrl_data;
unsigned int* test_speed;

void controller_init(quadcopter_intrinsics* qi,
        controller_data* cd,
        controller_parameters* cp,
        unsigned int* t)
{
    int i;

   //init the controller data storage
    quad_intrinsics = qi;
    ctrl_data = cd;
    controller_param = cp;
    test_speed = t;

    ctrl_data->U1 = 0;
    ctrl_data->U2 = 0;
    ctrl_data->U3 = 0;
    ctrl_data->U4 = 0;

    // init quadintrinsics data
    quad_intrinsics->IXX = 0.1296;
    quad_intrinsics->IYY = 0.1296;
    quad_intrinsics->IZZ = 0.2272;
    // total mass : 1800 g
    quad_intrinsics->m = 1.8;
    // Motor / Center distance: 32 cm
    quad_intrinsics->l = 0.32;
    quad_intrinsics->b = 0.538;
    quad_intrinsics->d = 0.011;
    quad_intrinsics->hl = 17.64;
    // Force needed to lift to quad: 17.64 N
```

```
    controller_param->FCS = 0;
    controller_param->auto_manual_sel = MANUAL_CTRL;
    controller_param->controller_counter = 0;
    controller_param->pitch_target = 0;
    controller_param->roll_target = 0;
    controller_param->yaw_target = 0;
    controller_param->manual_is_pressed = 0;

    for (i = 0; i < 4; i++) {
        test_speed[i] = ZERO_SPEED_THROTTLE;
    }
}

// PROCESS_COMMAND
//         switch the controller state according to input state
//         and reset the controller_counter. The point is to
//         achieve the state diagram for the controller(see doc)
//
// PARAM   byte_command  : command as requested by the PI
void process_command(unsigned char byte_instruction)
{
    // auto manual switch
    if (byte_instruction == BYTE_INSTRUCTION_AUTO_MANUAL) {
        if (controller_param->auto_manual_sel == AUTOPILOT) {
            printf("\n\rSwitch to manual control.\n\r");
            controller_param->auto_manual_sel = MANUAL_CTRL;
        } else {
            printf("\n\rSwitch to autopilot\n\r");
            controller_param->auto_manual_sel = AUTOPILOT;
            if (controller_param->FCS == FCS_OFFSET) {
                controller_param->FCS = FCS_HOVERING;
                controller_param->controller_counter = 0;
            }
        }
    // abort mission
    } else if (byte_instruction == BYTE_INSTRUCTION_ABORT){
        printf("\n\rAbort mission, start crash landing sequence\n\r");
        controller_param->FCS = FCS_CRASH;
        controller_param->controller_counter = 0;
    } else if (byte_instruction == BYTE_INSTRUCTION_PIC) {
        printf("\n\rCheese!\n\r");
    } else if (byte_instruction == BYTE_INSTRUCTION_VID) {
        printf("\n\The big brother is watching you!\n\r");
    } else if (byte_instruction == BYTE_INSTRUCTION_CLR) {
        printf("\n\rCleaning multimedia storage.\n\r");
    } else {
            // state FCS_PWRON
        if (controller_param->FCS == FCS_PWRON) {
            // power on to zero
            if (byte_instruction == BYTE_INSTRUCTION_ZERO) {
                controller_param->controller_counter = 0;
```

```
            controller_param->FCS = FCS_ZERO;
            printf("\n\rZeroing quadcopter\n\r");
        } else if (byte_instruction == BYTE_INSTRUCTION_MOTOR_TEST) {
            controller_param->FCS = FCS_MOTOR_TEST;
            controller_param->controller_counter = 0;
            printf("\n\rEnter motor testing mode\n\r");
        } else {
            error = INVALID_COMMAND;
        }
    // state FCS_ZREO
    } else if (controller_param->FCS == FCS_ZERO) {
        // rezero the quadcopter
        if (byte_instruction == BYTE_INSTRUCTION_ZERO) {
            controller_param->controller_counter = 0;
            controller_param->FCS = FCS_ZERO;
            printf("\n\rRezeroing quadcopter\n\r");
        // zero to take-off
        } else if (byte_instruction == BYTE_INSTRUCTION_TAKEOFF) {
            controller_param->controller_counter = 0;
            motor_on = 1;
            controller_param->FCS = FCS_TAKEOFF;
            printf("\n\rInit take-off sequence\n\r");
        } else {
            error = INVALID_COMMAND;
        }
    // state FCS_TAKEOFF
    } else if (controller_param->FCS == FCS_TAKEOFF) {
        // Ain't no command mess with take-off sequence. Ain't no command.
        //
http://static.fjcdn.com/pictures/Ain+t+no+one+fucks+with+tiny+hippo_7541d9_3298343.jpg
        error = INVALID_COMMAND;
    // state FCS_HOVERING
    } else if (controller_param->FCS == FCS_HOVERING) {
        if (byte_command == BYTE_INSTRUCTION_CONTROLLED_LANDING) {
            controller_param->controller_counter = 0;
            controller_param->FCS = FCS_LANDING;
            printf("\n\rInit landing sequence\n\r");
        } else {
            // Autopilot manuvers
            if (controller_param->auto_manual_sel == AUTOPILOT) {
                if (byte_instruction == BYTE_INSTRUCTION_W) {
                    controller_param->controller_counter = 0;
                    controller_param->FCS = FCS_AUTO_F;
                    printf("Autopilot moving forward.\n\r");
                } else if (byte_instruction == BYTE_INSTRUCTION_S) {
                    controller_param->controller_counter = 0;
                    controller_param->FCS = FCS_AUTO_B;
                    printf("Autopilot moving backward.\n\r");
                } else if (byte_instruction == BYTE_INSTRUCTION_A) {
                    controller_param->controller_counter = 0;
                    controller_param->FCS = FCS_AUTO_L;
```

```
            printf("Autopilot moving left.\n\r");
        } else if (byte_instruction == BYTE_INSTRUCTION_D) {
            controller_param->controller_counter = 0;
            controller_param->FCS = FCS_AUTO_R;
            printf("Autopilot moving right.\n\r");
        } else if (byte_instruction == BYTE_INSTRUCTION_Q) {
            controller_param->controller_counter = 0;
            controller_param->FCS = FCS_AUTO_YL;
            printf("Autopilot yaw left.\n\r");
        } else if (byte_instruction == BYTE_INSTRUCTION_E) {
            controller_param->controller_counter = 0;
            controller_param->FCS = FCS_AUTO_YR;
            printf("Autopilot yaw right.\n\r");
        } else if (byte_instruction == BYTE_INSTRUCTION_R) {
            controller_param->controller_counter = 0;
            controller_param->FCS = FCS_AUTO_U;
            printf("Autopilot elevation up.\n\r");
        } else if (byte_instruction == BYTE_INSTRUCTION_F) {
            controller_param->controller_counter = 0;
            controller_param->FCS = FCS_AUTO_D;
            printf("Autopilot elevation down.\n\r");
        } else {
        error = INVALID_COMMAND;
        }
    // Manual manuvers
    } else {
        if (byte_instruction == BYTE_INSTRUCTION_W ||
            byte_instruction == BYTE_INSTRUCTION_A ||
            byte_instruction == BYTE_INSTRUCTION_S ||
            byte_instruction == BYTE_INSTRUCTION_D ||
            byte_instruction == BYTE_INSTRUCTION_E ||
            byte_instruction == BYTE_INSTRUCTION_Q ||
            byte_instruction == BYTE_INSTRUCTION_R ||
            byte_instruction == BYTE_INSTRUCTION_F) {
            controller_param->controller_counter = 0;
            controller_param->FCS = FCS_OFFSET;
            controller_param->manual_is_pressed = 1;
            controller_param->manual_key_pressed = byte_instruction;
        } else {
            error = INVALID_COMMAND;
        }
    }
  }
// state FCS_OFFSET
} else if (controller_param->FCS == FCS_OFFSET){
  if (byte_command == BYTE_INSTRUCTION_AUTO_BALANCE) {
      controller_param->controller_counter = 0;
      controller_param->FCS = FCS_HOVERING;
      offset_clear();
      printf("Auto balance quadcopter.\n\r");
  } else if (byte_instruction == BYTE_INSTRUCTION_W ||
```

```
                byte_instruction == BYTE_INSTRUCTION_A ||
                byte_instruction == BYTE_INSTRUCTION_S ||
                byte_instruction == BYTE_INSTRUCTION_D ||
                byte_instruction == BYTE_INSTRUCTION_E ||
                byte_instruction == BYTE_INSTRUCTION_Q ||
                byte_instruction == BYTE_INSTRUCTION_R ||
                byte_instruction == BYTE_INSTRUCTION_F) {
            controller_param->manual_is_pressed = 1;
            controller_param->manual_key_pressed = byte_instruction;
          } else {
            error = INVALID_COMMAND;
          }
      // state FCS_LANDING
      } else if (controller_param->FCS == FCS_OFFSET){
          // Ain't no command mess with landing sequence. Ain't no command.
          //
http://static.fjcdn.com/pictures/Ain+t+no+one+fucks+with+tiny+hippo_7541d9_3298343.jpg
          error = INVALID_COMMAND;
      // state FCS_CRASH
      } else if (controller_param->FCS == FCS_CRASH){
          // Ain't no command mess with crash landing sequence. Ain't no command.
          //
http://static.fjcdn.com/pictures/Ain+t+no+one+fucks+with+tiny+hippo_7541d9_3298343.jpg
          error = INVALID_COMMAND;
      } else if (controller_param->FCS == FCS_MOTOR_TEST) {
          if (byte_instruction == BYTE_INSTRUCTION_W ||
                  byte_instruction == BYTE_INSTRUCTION_A ||
                  byte_instruction == BYTE_INSTRUCTION_S ||
                  byte_instruction == BYTE_INSTRUCTION_D ||
                  byte_instruction == BYTE_INSTRUCTION_E ||
                  byte_instruction == BYTE_INSTRUCTION_Q ||
                  byte_instruction == BYTE_INSTRUCTION_R ||
                  byte_instruction == BYTE_INSTRUCTION_F) {
            controller_param->manual_is_pressed = 1;
            motor_on = 1;
            controller_param->manual_key_pressed = byte_instruction;
          } else {
            error = INVALID_COMMAND;
          }
      }
   }
}

// TERMINATE_SEQUENCE
//       Terminate the current sequence and change states accordingly
//       The point is to achieve the state diagram for the controller(see doc)
void terminate_sequence(void)
{
   // terminating take-off sequence
   if (controller_param->FCS == FCS_TAKEOFF) {
      printf("Take-off Sequence complete\n\r");
```

```
        controller_param->FCS = FCS_HOVERING;
        controller_param->controller_counter = 0;
    // terminating crash-landing sequence
    } else if (controller_param->FCS == FCS_CRASH){
        printf("Crash landing sequence complete, good luck.\n\r");
        controller_param->FCS = FCS_PWRON;
        controller_param->controller_counter = 0;
    // terminating controlled-landing sequence
    } else if (controller_param->FCS == FCS_LANDING) {
        printf("Landing sequence complete.\n\r");
        controller_param->FCS = FCS_PWRON;
        controller_param->controller_counter = 0;
    // terminating autopilot manuvers
    } else if (controller_param->FCS == FCS_AUTO_F ||
                controller_param->FCS == FCS_AUTO_B ||
                controller_param->FCS == FCS_AUTO_L ||
                controller_param->FCS == FCS_AUTO_R ||
                controller_param->FCS == FCS_AUTO_U ||
                controller_param->FCS == FCS_AUTO_D ||
                controller_param->FCS == FCS_AUTO_YL||
                controller_param->FCS == FCS_AUTO_YR) {
        printf("Autopilot Maneuvering complete\n\r");
        controller_param->FCS = FCS_HOVERING;
        controller_param->controller_counter = 0;
    } else {
        // the current state is not terminable, cast error
        error = CANNOT_TERMINATE_SEQUENCE;
    }
}

// take down initial position of the quadcopter
// prepare for take off.
void zero_quadcopter(void)
{
    if (!controller_param->controller_counter) {
        // record the current pitch
        update_target_yaw(ctrl_data->yaw);

        printf("reset yaw: ");
        printFloat(controller_param->yaw_target);
        printf(" degrees \n\r");
        // zero the lift, yaw, pitch
        controller_param->lift = 0;
        offset_clear();
        controller_param->controller_counter = 1;
    }
}

// flight control routine that goes to the scheduler
unsigned int flight_control_routine(void)
{
```

```
if (controller_param->FCS == FCS_OFFSET) {
    if (controller_param->manual_is_pressed) {
        if (controller_param->manual_key_pressed == 'r') {
            lift_offset(1, LIFT_UNIT_CHANGE);
        } else if (controller_param->manual_key_pressed == 'f') {
            lift_offset(0, LIFT_UNIT_CHANGE);
        } else if (controller_param->manual_key_pressed == 'w') {
            pitch_offset(1, PITCH_UNIT_CHANGE);
        } else if (controller_param->manual_key_pressed == 's') {
            pitch_offset(0, PITCH_UNIT_CHANGE);
        } else if (controller_param->manual_key_pressed == 'a') {
            roll_offset(1, ROLL_UNIT_CHANGE);
        } else if (controller_param->manual_key_pressed == 'd') {
            roll_offset(0, ROLL_UNIT_CHANGE);
        } else if (controller_param->manual_key_pressed == 'q') {
            yaw_offset(1, YAW_UNIT_CHANGE);
        } else if (controller_param->manual_key_pressed == 'e') {
            yaw_offset(0, YAW_UNIT_CHANGE);
        } else if (controller_param->manual_key_pressed == ' ') {
            offset_clear();
        }
        //printf("%d.\n\r", ctrl_data->lift);
        controller_param->manual_is_pressed = 0;
    }
} else if (controller_param->FCS == FCS_TAKEOFF) {
    take_off();
} else if (controller_param->FCS == FCS_CRASH) {
    crash_landing();
} else if (controller_param->FCS == FCS_MOTOR_TEST) {
     if (controller_param->manual_is_pressed) {
        if (controller_param->manual_key_pressed == 'q') {
            motor_test(1, MOTOR_TEST_STEP, 1);
        } else if (controller_param->manual_key_pressed == 'a') {
            motor_test(0, MOTOR_TEST_STEP, 1);
        } else if (controller_param->manual_key_pressed == 'w') {
            motor_test(1, MOTOR_TEST_STEP, 2);
        } else if (controller_param->manual_key_pressed == 's') {
            motor_test(0, MOTOR_TEST_STEP, 2);
        } else if (controller_param->manual_key_pressed == 'e') {
            motor_test(1, MOTOR_TEST_STEP, 3);
        } else if (controller_param->manual_key_pressed == 'd') {
            motor_test(0, MOTOR_TEST_STEP, 3);
        } else if (controller_param->manual_key_pressed == 'r') {
            motor_test(1, MOTOR_TEST_STEP, 4);
        } else if (controller_param->manual_key_pressed == 'f') {
            motor_test(0, MOTOR_TEST_STEP, 4);
        }
        controller_param->manual_is_pressed = 0;
     }
} else if (controller_param->FCS == FCS_ZERO) {
    zero_quadcopter();
```

```
    } else if (controller_param->FCS == FCS_PWRON) {
        //TODO:
    } else if (controller_param->FCS == FCS_HOVERING) {`
        //TODO:
    } else if (controller_param->FCS == FCS_LANDING) {
        //TODO:
        controlled_landing();
    // AUTO PILOT CONTROLS
    } else if (controller_param->FCS == FCS_AUTO_F) {
        foward(1);
    } else if (controller_param->FCS == FCS_AUTO_B) {
        backward(1);
    } else if (controller_param->FCS == FCS_AUTO_L) {
        left(1);
    } else if (controller_param->FCS == FCS_AUTO_R) {
        right(1);
    } else if (controller_param->FCS == FCS_AUTO_U) {
        up(1);
    } else if (controller_param->FCS == FCS_AUTO_D) {
        down(1);
    } else if (controller_param->FCS == FCS_AUTO_YL) {
        yaw_CW(1);
    } else if (controller_param->FCS == FCS_AUTO_YR) {
        yaw_CCW(1);
    } else {
        error = 1;
    }
    // modify U1', U2', U3', U4' from the flight control routine
    return 1;
}

// Manual control
void offset_clear(void) {
    controller_param->pitch_target = 0;
    controller_param->roll_target  = 0;
}

// Increment or decrement the trottle value for all motors
void lift_offset(unsigned int sign, unsigned int value){
    if (sign) {
        update_lift(controller_param->lift + value);
    } else {
        update_lift(controller_param->lift - value);
    }
}

// increase or decrease the lift value for specific motor by a certain amount
void motor_test(unsigned int sign, unsigned int value, unsigned int motor_num) {
    unsigned int result;
    result = sign ? test_speed[motor_num-1] + value : test_speed[motor_num-1] - value;
    test_speed[motor_num-1] = result;
```

```
    printf("M%d: %u.\n\r", motor_num, result);
    setDutyCycle(result,motor_num);
}

// offset four motors
void roll_offset(unsigned int sign, unsigned int value) {
    if (sign) {
        update_target_pitch(controller_param->roll_target + (double)value);
    } else {
        update_target_pitch(controller_param->roll_target - (double)value);
    }
}

// offset the pitch
void pitch_offset(unsigned int sign, unsigned int value)
{
    if (sign) {
        update_target_pitch(controller_param->pitch_target + (double)value);
    } else {
        update_target_pitch(controller_param->pitch_target - (double)value);
    }
}

// offset the yaw
void yaw_offset(unsigned int sign, unsigned int value)
{
    if (sign) {
        update_target_yaw(controller_param->yaw_target + (double)value);
    } else {
        update_target_yaw(controller_param->yaw_target - (double)value);
    }
}

// Autopilot routines
void take_off(void)
{
    // TODO:
    // Slowly increase throttle to HOVERING_THROTTLE
    pid_clr();
    if (controller_param->controller_counter < 100) {
        motors_on(1);
        (controller_param->controller_counter)++;
    } else if (controller_param->controller_counter < 200 ){
        update_lift(10.0);
        (controller_param->controller_counter)++;
    } else if (controller_param->controller_counter < 300) {
        update_lift(20.0);
        (controller_param->controller_counter)++;
    } else if (controller_param->controller_counter < 400) {
        update_lift(30.0);
        (controller_param->controller_counter)++;
```

```
        // terminate the motors for debugging
    } else {
        terminate_sequence();
    }
}

// THE PANIC BUTTON
void crash_landing()
{
    // Temp crash landing solution:
    // Turn off the motors on the fly
    motors_on(0);
    test_speed[0] = ZERO_SPEED_THROTTLE;
    test_speed[1] = ZERO_SPEED_THROTTLE;
    test_speed[2] = ZERO_SPEED_THROTTLE;
    test_speed[3] = ZERO_SPEED_THROTTLE;
    terminate_sequence();
}

// safely land the aircraft
void controlled_landing()
{

}

// panorama maneuver
void panorama_maneuver()
{

}

// fly forward
void foward(unsigned int distance)
{

}

// fly backward
void backward(unsigned int distance)
{

}

// fly to the left
void left(unsigned int distance)
{

}

// fly to the right
void right(unsigned int distance)
```

```
{

}

void up(unsigned int distance)
{

}

void down(unsigned int distance)
{

}

void yaw_CW(unsigned int degree)
{

}

void yaw_CCW(unsigned int degree)
{

}

// return the new yaw
// update the yaw to the new yaw, report errors if limits are exceeded
void update_target_yaw(double new_yaw)
{
    double period;
    period = YAW_HIGH_BOND - YAW_LOW_BOND;
    while (new_yaw < YAW_LOW_BOND || new_yaw > YAW_HIGH_BOND) {
        if (new_yaw < YAW_LOW_BOND) {
            new_yaw + period;
        } else {
            new_yaw - period;
        }
    }
    controller_param->yaw_target =  new_yaw;
}


// return the new yaw
// update the yaw to the new yaw, report errors if limits are exceeded
void update_target_pitch(double new_pitch)
{
    double period;
    period = PITCH_HIGH_BOND - PITCH_LOW_BOND;
    if (new_pitch < PITCH_LOW_BOND) {
        controller_param->pitch_target = PITCH_LOW_BOND;
    } else if (new_pitch > PITCH_HIGH_BOND) {
        controller_param->pitch_target = PITCH_HIGH_BOND;
```

```
    } else {
        controller_param->pitch_target = new_pitch;
    }
}


// return the new yaw
// update the yaw to the new yaw, report errors if limits are exceeded
void update_target_roll(double new_roll)
{
    double period;
    period = ROLL_HIGH_BOND - ROLL_LOW_BOND;
    if (new_roll < ROLL_LOW_BOND) {
        controller_param->roll_target = ROLL_LOW_BOND;
    } else if (new_roll > ROLL_HIGH_BOND) {
        controller_param->roll_target = ROLL_HIGH_BOND;
    } else {
        controller_param->roll_target = new_roll;
    }
}

void update_lift(double new_lift)
{
    if (new_lift >= MAX_LIFT) {
        controller_param->lift = MAX_LIFT;
    } else if (new_lift < 0){
        controller_param->lift = 0;
    } else {
        controller_param->lift = new_lift;
    }
}
```

*scheduler.h*

```
// schedule the sensor module to achieve a certain sampling rate and low power consumption
#ifndef RESET_H
#define RESET_H

#include "routines.h"

#define TOTAL_TICKS 1000

void timer_init(void);
void start_scheduler(void);
void update_status(int status);
int check_status(void);
void timer_rst(void);

extern unsigned int done;
extern unsigned int curr_channel;
```

```
extern unsigned int freqCounter;

#endif
```

*scheduler.c*

```
#include "scheduler.h"

unsigned int done;
unsigned int motorAll, high1, high2, high3, high4;


void timer_init(void) {
    // Timer2, used as the scheduler timer
    // choose a prescale of 4
    T2CONbits.T2CKPS = 0b01;
    // set period for the scheduler counter
    PR2 = 165;
    // turn on Timer2
    done = 0;
    curr_channel = 0;
    T2CONbits.TMR2ON = 1;
}

void start_scheduler(void) {
    while (1) {

        //timer interrupt
        if (sampling_flag == 1) {
            timer_rst();
            //settlingTime++;
            if (curr_channel == 1) {
                update_status(SYSTEM_MOTOR);
                PORTAbits.RA5 = !PORTAbits.RA5;
                PORTB = 0xFF;
                start_motor_timers();
            }

            if (curr_channel == 100) {
                update_status(SYSTEM_I2C_R);
                //done = read_sensor_dummy();
                 done = read_sensor();
            }

            if (curr_channel == 400) {
                update_status(SYSTEM_CONTROLLER);
                done = flight_control_routine();
            }
            if (curr_channel == 500){
                update_status(SYSTEM_STABILIZER);
                done = stabilize_routine();
```

```
            }

            if (curr_channel == 600) {
                update_status(SYSTEM_UART);
                done = uart_task();
            }

            if (curr_channel == 700) {
                update_status(SYSTEM_MOTOR);
                done = controlled_speed();
            }


            //update status if a task terminate
        } else {
            if ((check_status() == SYSTEM_UART) && done && TXSTAbits.TRMT) {
                update_status(SYSTEM_IDLE);
                done = 0;
            } else if ( done && (check_status() == SYSTEM_I2C_R ||
                        check_status() == SYSTEM_STABILIZER ||
                        check_status() == SYSTEM_CONTROLLER)) {
                update_status(SYSTEM_IDLE);
                done = 0;
            }
        }
    }
}

// update system status to R5 RC2 RC1 RC0

void update_status(int status) {
    int status_out = 0;
    status_out = status_out | (status & 0x01) | (status & 0x02) | status & 0x04 | ((status & 0x08)
<< 2);
    PORTC = status_out;
}

int check_status(void) {
    return PORTCbits.RC0
            | (PORTCbits.RC1 << 1)
            | (PORTCbits.RC2 << 2)
            | (PORTCbits.RC5 << 5);
}

void timer_rst(void) {
    sampling_flag = 0;
    curr_channel = (curr_channel >= TOTAL_TICKS) ? 0: curr_channel+1;
}

// delay n clk_cycle, update curr_chan;
```

```
void timer0_delay(int clk_cycle) {
    int count = 0;
    while (count < clk_cycle) {
        if (sampling_flag) {
            timer_rst();
            count++;
        }
    }
}
```

*stabilizer.h*

```
#ifndef STABILIZER_H
#define STABILIZER_H

#include "routines.h"

/*****************************************************
*           EE478 quadcopter stabilizer             *
* The motor speeds parameters are stabilized here   *
*****************************************************/

typedef struct
{
    double roll_error;
    double pitch_error;
    double yaw_error;
    double altitude_error;
} kinetic_errors;

typedef struct
{
    double en0;
    double en1;
    double en2;        // error[n], error[n-1], error[n-2]
    double fn0;
    double fn1;
    double fn2;        // output[n-1], output[n-1], output[n-2]
} PID_storage  ;

extern kinetic_errors* kinetic_errs;
extern PID_storage* roll_PID;
extern PID_storage* pitch_PID;
extern PID_storage* yaw_PID;

void stabilize_init(kinetic_errors*, PID_storage*,  PID_storage*, PID_storage*);
unsigned int stablize_routine(void);
unsigned char roll_stablize(void);
unsigned char pitch_stablize (void);
unsigned char yaw_stablize (void);
unsigned char altitude_stabilize (void);
```

```
void update_errors(void);
void inverted_motion_matrix(void);
unsigned int stabilize_routine(void);
void pid_clr(void);
#endif
```

*stabilizer.c*

```
#include "stabilizer.h""

kinetic_errors* kinetic_errs;
PID_storage* roll_PID;
PID_storage* pitch_PID;
PID_storage* yaw_PID;

void stabilize_init(kinetic_errors* kinetic_errors_ptr, PID_storage* roll,
                PID_storage* pitch, PID_storage* yaw)
{
   kinetic_errs = kinetic_errors_ptr;
   // initialize structs
   kinetic_errs->altitude_error = 0;
   kinetic_errs->pitch_error    = 0;
   kinetic_errs->roll_error     = 0;
   kinetic_errs->yaw_error      = 0;

   // initilize PIDs
   roll_PID = roll;
   pitch_PID = pitch;
   yaw_PID = yaw;
   pid_clr();
}

void pid_clr(void)
{
   roll_PID->en0 = 0.0;
   roll_PID->en1 = 0.0;
   roll_PID->en2 = 0.0;
   roll_PID->fn0 = 0.0;
   roll_PID->fn1 = 0.0;
   roll_PID->fn2 = 0.0;

   pitch_PID->en0 = 0.0;
   pitch_PID->en1 = 0.0;
   pitch_PID->en2 = 0.0;
   pitch_PID->fn0 = 0.0;
   pitch_PID->fn1 = 0.0;
   pitch_PID->fn2 = 0.0;

   yaw_PID->en0 = 0.0;
   yaw_PID->en1 = 0.0;
```

```
    yaw_PID->en2 = 0.0;
    yaw_PID->fn0 = 0.0;
    yaw_PID->fn1 = 0.0;
    yaw_PID->fn2 = 0.0;
}

// lower level stabilizers to approximate hovering state
unsigned int stabilize_routine(void)
{
    update_errors();
    roll_stablize();
    pitch_stablize();
    altitude_stabilize();
    yaw_stablize();
    inverted_motion_matrix();
    return 1;
}

unsigned char altitude_stabilize(void)
{
    // TODO: make the altitude controlled
    ctrl_data->U1 = controller_param->lift;
}

unsigned char roll_stablize(void)
{
    roll_PID->en0 = kinetic_errs->roll_error;
    // compute next output
    roll_PID->fn0 = 1.055* roll_PID->en0 - 1.007*roll_PID->en1 + 0.7917*roll_PID->fn1;
    ctrl_data->U2 = roll_PID->fn0;
    // store prev data
    roll_PID->en2 = roll_PID->en1;
    roll_PID->en1 = roll_PID->en0;
    roll_PID->fn2 = roll_PID->fn1;
    roll_PID->fn1 = roll_PID->fn0;
    return 1;
}

unsigned char pitch_stablize (void)
{
    pitch_PID->en0 = (double)kinetic_errs->pitch_error;
    // compute next output
    roll_PID->fn0 = 1.055* roll_PID->en0 - 1.007*roll_PID->en1 + 0.7917*roll_PID->fn1;
    ctrl_data->U3 = pitch_PID->fn0;
    // store prev data
    pitch_PID->en2 = pitch_PID->en1;
    pitch_PID->en1 = pitch_PID->en0;
    pitch_PID->fn2 = pitch_PID->fn1;
    pitch_PID->fn1 = pitch_PID->fn0;
    return 1;
}
```

```
unsigned char yaw_stablize (void)
{
   yaw_PID->en0 = (double)kinetic_errs->yaw_error;
   // compute next output
   roll_PID->fn0 = 1.105* roll_PID->en0 - 1.07*roll_PID->en1 + 0.8535*roll_PID->fn1;
   ctrl_data->U4 = yaw_PID->fn0;
   // store prev data
   yaw_PID->en2 = yaw_PID->en1;
   yaw_PID->en1 = yaw_PID->en0;
   yaw_PID->fn2 = yaw_PID->fn1;
   yaw_PID->fn1 = yaw_PID->fn0;
   return 1;
}

// compute the system error
void update_errors(void)
{
   kinetic_errs->altitude_error = 0;
   kinetic_errs->pitch_error = controller_param->pitch_target - ctrl_data->pitch;
   kinetic_errs->roll_error = controller_param->roll_target - ctrl_data->roll;
   kinetic_errs->yaw_error = controller_param->yaw_target - ctrl_data->yaw;
}

// inverted motion matrix operation
void inverted_motion_matrix(void)
{
   double temp1, temp2, temp3, temp4, temp5;
   temp5 = (quad_intrinsics->l*quad_intrinsics->b);
   temp1 = ctrl_data->U1/4/quad_intrinsics->b;
   temp2 = ctrl_data->U2/2/temp5;
   temp3 = ctrl_data->U3/2/temp5;
   temp4 = ctrl_data->U4/4/quad_intrinsics->d;

   ctrl_data->omega1square = temp1 - temp3 - temp4;
   ctrl_data->omega2square = temp1 - temp2 + temp4;
   ctrl_data->omega3square = temp1 + temp3 - temp4;
   ctrl_data->omega4square = temp1 + temp2 + temp4;
}
```

*uart.h*

```
#ifndef UART_H
#define UART_H

#define BUFFER_SIZE 1

// REPORT_FLAG indicate what the terminal
// is going to output

// The terminal sends the welcome msg
```

```
#define REPORT_FLAG_ECHO                1

#define BYTE_INSTRUCTION_ZERO           'z'
#define BYTE_INSTRUCTION_TAKEOFF        't'
#define BYTE_INSTRUCTION_CONTROLLED_LANDING 'l'
#define BYTE_INSTRUCTION_ABORT          'c'
#define BYTE_INSTRUCTION_AUTO_MANUAL    'x'
#define BYTE_INSTRUCTION_AUTO_BALANCE   ' '
#define BYTE_INSTRUCTION_W              'w'
#define BYTE_INSTRUCTION_A              'a'
#define BYTE_INSTRUCTION_S              's'
#define BYTE_INSTRUCTION_D              'd'
#define BYTE_INSTRUCTION_Q              'q'
#define BYTE_INSTRUCTION_E              'e'
#define BYTE_INSTRUCTION_R              'r'
#define BYTE_INSTRUCTION_F              'f'
#define BYTE_INSTRUCTION_SYSINFO        'i'
#define BYTE_INSTRUCTION_MOTOR_TEST     'm'
#define BYTE_INSTRUCTION_PIC            'p'
#define BYTE_INSTRUCTION_VID            'v'
#define BYTE_INSTRUCTION_CLR            'u'


#include "routines.h"

typedef struct
{
   unsigned char overflow;
   char sign;
   unsigned int decimal_left;
   unsigned int decimal_right;
} double_printable;

//global variables
extern char* RX_buffer_ptr;
extern unsigned int report_flag;
extern volatile int remaining_buffer_size;
extern volatile unsigned int RX_buffer_full;
extern volatile unsigned char byte_command;
extern unsigned int report_type;

// public functions
void uart_init(void);
void baud_test(void);
int uart_task(void);
void RX_ISR(void);
void RX_command_process(void);
void printFloat(double fInput);

// private functions
void RX_buffer_push(char RX_buffer_push);
```

```
void _user_putc(char TX_byte);
void RX_open(void);
void RX_close(void);
unsigned int verify_data(char* data_ptr);


extern int count_sensor;

#endif
```

*uart.c*

```
#include "uart.h"

// UART high-level protocols:

// modifying flight control signal flow according to
// the task given
// Echo that command is received
int uart_task(void)
{
    char test = 0x55;
    //baud_test();
    if (RX_buffer_full) {
        RX_command_process();
        RX_buffer_full = 0;
    }

    if (error == NO_ERROR) {
        //printf("\n\rFCS: %d\n\r", controller_param->FCS);
       //printf("%li, %li, %li, %li\n\r", ctrl_data->roll, ctrl_data->pitch, ctrl_data->yaw, ctrl_data-
>altitude);


    } else if (error == SERIAL_TRANSMISSION_ERROR) {


        printf("\n\rSERIAL_ERROR\n\r");
                    error = NO_ERROR;
    } else if (error == I2C_TRANSMISSION_ERROR) {
        printf("\n\rI2C_ERROR\n\r");
                    error = NO_ERROR;
    } else if (error == INVALID_COMMAND) {
            printf("\n\rInvalid command\n\r");
            error = NO_ERROR;
            }

    return 1;
}

//transmit 0x55 to test baud rate of TX
```

```
void baud_test(void)
{
        char test_byte = 't';
        _user_putc(test_byte);
}

void RX_command_process(void) {
   int i;
   unsigned char command;
   // pharsing the command
   for (i = 0; i < BUFFER_SIZE; i++) {
      command = (unsigned char)byte_command;
      process_command(command);
   }
   RX_open();
}

//Low level protocols

//enable the receiver
void RX_open(void) {
   RCSTA1bits.CREN1 = 1;
   PIE1bits.RC1IE = 1;
   PIR1bits.RC1IF = 0;
   remaining_buffer_size = BUFFER_SIZE;
}

// close RX port when there are command to be executed
void RX_close(void) {
   RCSTA1bits.CREN1 = 0;
   PIE1bits.RC1IE = 0;
}

// pull the a byte out of the RX register and append it
// to the command string if there is no frame error
void RX_buffer_push(char data_byte) {
   *(RX_buffer_ptr + BUFFER_SIZE - remaining_buffer_size) = data_byte;
   remaining_buffer_size--;
   if (!remaining_buffer_size) {
      RX_close();
   }
}

void uart_init(void) {
        // Asynchronous mode, high speed, 9th bit-disabled
        TXSTA1 =   0b10100101;
        // 8-bit reception, clear framing and overrun error
        RCSTA1 =   0b10011000;
        // TX/RX high true. 1 as idle bit. 8-bit baud
        // rate generator, auto baud disabled
        BAUDCON1 = 0b01000000;
```

```
        SPBRGH1= 0x00;
        SPBRG1 = 0xCF;
    RX_open();
    byte_command = '0';
}

// send out a byte via uart or LCD, used as stdoutput
void _user_putc(char TX_byte)
{
        // turning on transmitting LED
        while(!TXSTAbits.TRMT) {
        if (sampling_flag) {
            timer_rst();
        }
    }
        TXREG1 = TX_byte;
}

void printFloat(double fInput)
{
   //The number is converted to two parts.
   long lWhole=(long)((double)fInput);
   unsigned long ulPart=(unsigned long)((double)fInput*100)-lWhole*100;

   printf(" %li.%lu",lWhole,ulPart);

}
```

*interrupt.h*

```
#ifndef INT_H
#define INT_H

#include "routines.h"

void interrupt_init(void);
void global_interrupt(void);
void myISR(void);

#endif
```

*interrupt.c*

```
#include "interrupts.h"

void interrupt_init(void)
{
   // disable all interrupts except for external interrupt on INT2/RA2
   // and timer0/1 interrupt
   // disable interrupt priority
   RCONbits.IPEN = 1;
```

```
    INTCON =  0b00100000;       // timer0 overflow interrupt
    INTCON2 = 0b11000100;  // Pull-ups disabled/Interrupt0 Rising edge
    INTCON3 = 0b00000000;  // external to zero

    PIE1bits.RC1IE = 1;   // serial RX interrupt
    PIE1bits.TMR2IE = 1;  // enable timer2 interrupt

    PIE1bits.TMR1IE = 1;  // enable timer1 interrupt
    PIE2bits.TMR3IE = 1;  // enable timer3 interrupt
    PIE5bits.TMR5IE = 1;  // enable timer5 interrupt
    INTCONbits.T0IE = 1;  // enable timer0 interrupt

    IPR1bits.RC1IP = 1;
    IPR1bits.TMR2IP = 1;
    IPR2bits.TMR3IP = 1;
    IPR1bits.TMR1IP = 1;
    IPR5bits.TMR5IP = 1;

    INTCONbits.GIE_GIEH = 1;          /* Enable the interrupts */
    INTCONbits.PEIE_GIEL = 1;         /* Enable peripheral interrupts */
}

#pragma code high_vector=0x08
void global_interrupt(void) {
    _asm GOTO myISR _endasm;
}
#pragma code

#pragma interrupt myISR
void myISR(void)
{
    // check frame error
    if (PIR1bits.TMR2IF) {
        // restart the counter
        TMR2 = 0;
        PIR1bits.TMR2IF = 0;
        sampling_flag = 1;
    }

    if (PIR1bits.RC1IF) {
      PIR1bits.RC1IF = 0;
      byte_command = RCREG1;
      byte_command = RCREG1;

      remaining_buffer_size--;
      if (!remaining_buffer_size) {
            RX_buffer_full = 1;
            RCSTA1bits.CREN1 = 0;
            PIE1bits.RC1IE = 0;
      }
      INTCONbits.GIE = 1;
```

```
    }

    if (INTCONbits.T0IF) {
        TMR0H:TMR0L = 0;
        INTCONbits.TMR0IF = 0;
        T0CONbits.TMR0ON = 0;
        PORTBbits.RB0 = 0;
    }

    if (PIR1bits.TMR1IF) {
        TMR1H:TMR1L = 0;
        PIR1bits.TMR1IF = 0;
        T1CONbits.TMR1ON = 0;
        PORTBbits.RB5 = 0;
    }

    if (PIR2bits.TMR3IF) {
        TMR3H:TMR3L = 0;
        PIR2bits.TMR3IF = 0;
        T3CONbits.TMR3ON = 0;
        PORTBbits.RB2 = 0;
    }

    if (PIR5bits.TMR5IF) {
        TMR5H:TMR5L = 0;
        PIR5bits.TMR5IF = 0;
        T5CONbits.TMR5ON = 0;
        PORTBbits.RB3 = 0;
    }
}
```

*intracomm.h*

```
#ifndef INTRA_H
#define INTRA_H

#include "routines.h"

#define TIMEOUT 35
#define SENSOR_CARD_ADDR 0x55
#define THROUGHPUT 16

extern char* I2C_TX_buffer_ptr;
extern char* I2C_RX_buffer_ptr;

// low level I2C functions
void I2C_master_init(char* TX_buffer, char* RX_buffer);
void master_update_bit_field(unsigned int device_ID, unsigned char sub_addr,
                        unsigned char bit_field, unsigned char mask);
void master_I2C_read(unsigned int device_ID, int sub_addr);
void master_I2C_write(unsigned int device_ID, int registerAddr);
```

```
unsigned char master_I2C_read_byte(unsigned int device_ID, int registerAddr);
void master_I2C_write_byte(unsigned int device_ID, int registerAddr,
                    unsigned char data);
void c_WriteI2C( unsigned char data_out );
unsigned char c_ReadI2C( void );

// high level I2C protocols
void I2C_test(char* buffer);
unsigned int read_sensor(void);
unsigned int read_sensor_dummy(void);

void c_StartI2C(void);
// private functions
unsigned int mask2shift(unsigned char mask);
void c_NackI2C( void );
void c_RestartI2C(void);
void c_ackI2C(void);
void c_stopI2C(void);
void c_idleI2C(void);

#endif
```

*intracomm.h*

```
#ifndef INTRA_H
#define INTRA_H

#include "routines.h"

#define TIMEOUT 35
#define SENSOR_CARD_ADDR 0x55
#define THROUGHPUT 16

extern char* I2C_TX_buffer_ptr;
extern char* I2C_RX_buffer_ptr;

// low level I2C functions
void I2C_master_init(char* TX_buffer, char* RX_buffer);
void master_update_bit_field(unsigned int device_ID, unsigned char sub_addr,
                     unsigned char bit_field, unsigned char mask);
void master_I2C_read(unsigned int device_ID, int sub_addr);
void master_I2C_write(unsigned int device_ID, int registerAddr);

unsigned char master_I2C_read_byte(unsigned int device_ID, int registerAddr);
void master_I2C_write_byte(unsigned int device_ID, int registerAddr,
                    unsigned char data);
void c_WriteI2C( unsigned char data_out );
unsigned char c_ReadI2C( void );

// high level I2C protocols
```

```
void I2C_test(char* buffer);
unsigned int read_sensor(void);
unsigned int read_sensor_dummy(void);

void c_StartI2C(void);
// private functions
unsigned int mask2shift(unsigned char mask);
void c_NackI2C( void );
void c_RestartI2C(void);
void c_ackI2C(void);
void c_stopI2C(void);
void c_idleI2C(void);

#endif
```

*Intracomm.c*

```
#include "intra_comm.h"
#include "i2c.h"

char* I2C_TX_buffer_ptr;
char* I2C_RX_buffer_ptr;
unsigned int TX_buffer_size;
unsigned int RX_buffer_size;
unsigned int secret_code;

/*****************************************************
 Low-level intrasystem comm protocols
 *****************************************************/

void I2C_master_init(char* TX_buffer, char* RX_buffer){
    I2C_TX_buffer_ptr = TX_buffer;
    I2C_RX_buffer_ptr = RX_buffer;

    // Initiate registers related to SSP pheripheral in the PIC
    // FOSC = 64MHz, I2C clock = 100kHz
    SSP1ADDbits.SSP1ADD = 0x9F;
    // slew rate control desabled for 100kHz
    SSP1STATbits.SMP = 0;
    // disable SMbus input level
    SSP1STATbits.CKE = 0;
    // set RC4 and RC3 as open drain I2C ports
    SSP1CON1bits.SSPEN = 1;
    // I2C master mode, clock = FOSC/(4*(SSP1ADD+1))
    SSP1CON1bits.SSPM = 0b1000;
    // enable receive mode for I2C master
    SSP1MSK = 0xFF;
    // enable general call
    SSP1CON2bits.GCEN = 1;
    // transmit mode
    SSP1CON2bits.RCEN = 0;
```

```
    // disable data/addr hold
    SSP1CON3bits.AHEN = 0;
    SSP1CON3bits.DHEN = 0;
    // enable start/stop condition interrupt enable
    SSP1CON3bits.PCIE = 1;
    SSP1CON3bits.SCIE = 1;
    // 100ns hold time
    SSP1CON3bits.SDAHT = 0;
    // disable bus collision
    SSP1CON3bits.SBCDE = 0;
}

// MASTER_I2C_WRITE_BYTE
//        write a byte to a specific location of a slave device
//
// PARAM   device_ID   : I2C address of the slave
//        registerAddr: register address for a specific variable
//        data        : the data type being written.
void master_update_bit_field(unsigned int device_ID, unsigned char sub_addr,
                        unsigned char bit_field, unsigned char mask)
{
    unsigned char temp;
    temp = master_I2C_read_byte(device_ID, sub_addr);
    temp = (temp & ~mask) | (bit_field << (mask2shift(mask)) & mask);
    master_I2C_write_byte(device_ID, sub_addr, temp);
}

unsigned int mask2shift(unsigned char mask)
{
    unsigned int count;
    unsigned char last_bit_mask;
    count = 0;
    last_bit_mask = 0x01;
    while(!(mask & last_bit_mask)) {
        mask = mask >> 1;
        count++;
    }
    return count;
}

// MASTER_I2C_READ_BYTE
//        wait for device to reply, read a byte into the first
//        byte of the data buffer.
//
// RETURNS the byte being read as unsigned char.
// PARAM   device_ID:    I2C address of the slave
//        registerAddr: register address of the
//        variable being read.
unsigned char master_I2C_read_byte(unsigned int device_ID, int registerAddr)
{
```

```
    RX_buffer_size = 1;
    master_I2C_read(device_ID, registerAddr);
    RX_buffer_size = 0;
    return I2C_RX_buffer_ptr[0];
}

// MASTER_I2C_WRITE_BYTE
//        write a byte to a specific location of a slave device
//
// PARAM   device_ID   : I2C address of the slave
//        registerAddr: register address for a specific variable
//        data        : the data type being written.
void master_I2C_write_byte(unsigned int device_ID, int registerAddr,
                           unsigned char data)
{
    TX_buffer_size = 1;
    I2C_TX_buffer_ptr[0] = data;
    master_I2C_write(device_ID, registerAddr);
    TX_buffer_size = 0;
}

// MASTER_I2C_WRITE
//        write whatever's inside the TX_buffer to the slave device
//
// PARAM   device_ID   : I2C address of the slave
//        registerAddr: register address for a specific variable
void master_I2C_write(unsigned int device_ID, int registerAddr)
{
    int i;
    SSP1CON2bits.RCEN = 0;

    IdleI2C();                      // Wait until the bus is idle
    StartI2C();                      // Send START condition
    IdleI2C();                      // Wait for the end of the START condition
    WriteI2C( device_ID << 1 | 0x00 );  // Send address with R/W cleared for write
    IdleI2C();                      // Wait for ACK
    check_ack();

    WriteI2C(registerAddr);              // Write first byte of data
    IdleI2C();  // Wait for ACK
    check_ack();

    for (i = 0; i < TX_buffer_size; i++)
    {
        WriteI2C(I2C_TX_buffer_ptr[i]);              // Write first byte of data
            IdleI2C();  // Wait for ACK
        check_ack();
    }
    StopI2C();                       // Hang up, send STOP condition
}
```

```
// MASTER_I2C_READ
//        read an array of data from the save device to the data_buffer
//
// PARAM   device_ID   : I2C address of the slave
//        registerAddr: register address for a specific variable
//        size        : size of the array being read
//void master_I2C_read(unsigned int device_ID, int registerAddr)
//{
//    // write addr and subaddress
//    int i;
//    //SSP2CON2bits.RCEN = 1;
//
//    IdleI2C();                      // Wait until the bus is idle
//    StartI2C();                      // Send START condition
//    IdleI2C();                      // Wait for the end of the START condition
//    c_WriteI2C( device_ID <<1 | 0x00 );  // Send address with R/W cleared for write
//    IdleI2C();                      // Wait for ACK
//    check_ack();
//
//    WriteI2C(registerAddr);              // Write first byte of data
//    IdleI2C();  // Wait for ACK
//    check_ack();
//
//    //restart conditinon
//    c_RestartI2C();
//
//    //receive reading
//    IdleI2C();                      // Wait for the end of the START condition
//    WriteI2C( device_ID << 1 | 0x01 ); // Send address with R/W set for read
//    IdleI2C();
//    check_ack();
//
//    // wait for data sent and acked
//    for(i=0; i< RX_buffer_size-1; i++)
//    {
//         I2C_RX_buffer_ptr[i] = c_ReadI2C();
//        c_ackI2C();       //sec_ackndc_ack ACK
//    }
//    I2C_RX_buffer_ptr[RX_buffer_size - 1] = c_ReadI2C(); // Read nth byte of data
//    c_NackI2C();                    // Send NACK
//    c_stopI2C();                     // Hang up, send STOP condition
//}


// wait for device to reply, and read the data into RX buffer
void master_I2C_read(unsigned int device_ID, int registerAddr)
{
    int i;
    IdleI2C();
    c_StartI2C();                    // Send START condition
```

```
    IdleI2C();                  // Wait for the end of the START condition
    c_WriteI2C( device_ID << 1 | 0x01 ); // Send address with R/W set for read
    IdleI2C();
    check_ack();
    // wait for data sent and acked
    for(i=0; i< RX_buffer_size-1; i++)
    {
          I2C_RX_buffer_ptr[i] = c_ReadI2C();
        // Read first byte of data
         c_ackI2C();       //send ACK
         IdleI2C();
    }
    I2C_RX_buffer_ptr[RX_buffer_size-1] = c_ReadI2C();              // Read nth byte of data
    NotAckI2C();                    // Send NACK
    c_stopI2C();                     // Hang up, send STOP condition
}

// wait until ack arrived or ack timeout
void check_ack(void)
{
    int time_out = 0;
    int ack = SSPCON2bits.ACKSTAT;
    while (ack && time_out < TIMEOUT) {
        ack = SSPCON2bits.ACKSTAT;
        if (sampling_flag) {
            timer_rst();
            time_out++;
        }
    }
}

unsigned char c_ReadI2C( void )
{
    int time_out = 0;
    unsigned char result;
    SSP1CON2bits.RCEN = 1;
    while ( !SSP1STATbits.BF && time_out < TIMEOUT ){
        if (sampling_flag) {
            timer_rst();
            time_out++;
        }
    }     // wait until byte received
    if (time_out >= TIMEOUT) {
        error = I2C_TRANSMISSION_ERROR;
    }
    result = SSP1BUF;
    return result;              // return with read byte
}

// write to I2C sensor card
void c_WriteI2C( unsigned char data_out )
```

```
{
    int time_out = 0;
    SSP1BUF = data_out;           // write single byte to SSPBUF
    if ( SSP1CON1bits.WCOL) {     // test if write collision occurred
        printf("\n\r Error: Write I2C timeout.");
    } else {
        while( SSP1STATbits.BF && time_out < TIMEOUT);   // wait until write cycle is complete
            if (sampling_flag) {
                timer_rst();
                time_out++;
            }
        if (time_out >= TIMEOUT) {
            printf("\n\r Error: Write I2C timeout.");
        }
    }
}

// somehow not functinal
void c_NackI2C( void )
{
  SSP1CON2bits.ACKDT = 1;         // set acknowledge bit for not ACK
  SSP1CON2bits.ACKEN = 1;         // initiate bus acknowledge sequence
}

//ack the I2C
void c_ackI2C(void)
{
    SSP1CON2bits.ACKDT=0;
    SSP1CON2bits.ACKEN=1;
}

void c_stopI2C(void)
{
    SSP1CON2bits.PEN = 1;
}

void c_RestartI2C( void )
{
  SSP1CON2bits.RSEN = 1;          // initiate bus restart condition
}

//start I2C
void c_StartI2C(void)
{
    SSP1CON2bits.SEN = 1;
}

void c_idleI2C(void)
{
  while ( ( SSP1CON2 & 0x1F ) || ( SSPSTATbits.R_W ) )
     if (sampling_flag) {
```

```
        timer_rst();
    }
}

/*****************************************************
High-level intrasystem comm protocols
 *****************************************************/
// functions for masters

//void I2C_test(char* buffer_ptr)
//{
//    buffer_ptr = I2C_TX_buffer_ptr;
//    //send hello to device 0xFF at 0xFF
//    //sprintf(I2C_TX_buffer_ptr, "Hello");
//    buffer_ptr[0] = 'H';
//    buffer_ptr[1] = 'e';
//    buffer_ptr[2] = 'l';
//    buffer_ptr[3] = 'l';
//    buffer_ptr[4] = 'o';
//    TX_buffer_size = 5;
//    master_I2C_write(0x55, 0x33);
//}

// read dummy sensor data for testing purpose
unsigned int read_sensor_dummy(void)
{
    ctrl_data->roll = 16.25;
    ctrl_data->pitch = 0;
    ctrl_data->yaw   = 0;
    ctrl_data->altitude = 0;
}

// read the roll, pitch, yaw and height from the sensor card
unsigned int read_sensor(void)
{
    long temp;
    // use multiple bytes read
    RX_buffer_size = THROUGHPUT;

    master_I2C_read(0x55, secret_code);

    temp = 0;
    temp =  I2C_RX_buffer_ptr[1];
    temp =  (temp << 8) | I2C_RX_buffer_ptr[2];
    temp =  (temp << 8) | I2C_RX_buffer_ptr[3];
    temp =  (temp << 8) | I2C_RX_buffer_ptr[4];
    ctrl_data->roll = ((double)temp)/1000;

    temp    =  I2C_RX_buffer_ptr[5];
    temp    =  (temp << 8) | I2C_RX_buffer_ptr[6];
    temp    =  (temp << 8) | I2C_RX_buffer_ptr[7];
```

```
    temp    =  (temp << 8) | I2C_RX_buffer_ptr[8];
    ctrl_data->pitch = ((double)temp)/1000;

    temp     =  I2C_RX_buffer_ptr[9];
    temp     =  (temp) | I2C_RX_buffer_ptr[10];
    temp     =  (temp) | I2C_RX_buffer_ptr[11];
    temp     =  (temp) | I2C_RX_buffer_ptr[12];
    ctrl_data->yaw = ((double)temp)/1000;

    temp =  I2C_RX_buffer_ptr[13];
    temp =  (temp) | I2C_RX_buffer_ptr[14];
    temp =  (temp) | I2C_RX_buffer_ptr[15];
    temp =  (temp) | I2C_RX_buffer_ptr[16];
    ctrl_data->altitude = ((double)temp)/1000;

    RX_buffer_size = 0;
    return 1;

}

// check the first integer of RX_buffer
// return 1 if data is valid
unsigned int verify_data(char* buffer)
{
    unsigned int code;
    unsigned int result;
    code = bytes2int(buffer[0], buffer[1]);
    //printf("\n\rsecret is %d\n\r", secret_code);
    //update secret code
    result = (code == secret_code) ? 1:0;
    secret_code = (secret_code + 271) % 51;
    return result;
}
```

*motor_drive.h*

```
#ifndef MOTOR_DRIVE_H
#define          MOTOR_DRIVE_H
#include "routines.h"

#define SPEED_REPORT_RATE 60
#define THROTTLE_LOW 4000
#define ZERO_SPEED_THROTTLE 700 // throttle that just enough to keep to motors still
#define MAX_THROTTLE 999999999
#define SPEED2THROTTLE_FACTOR 1
```

```
#define MOTOR_TIMER_PRESCALE 0b10

extern unsigned int motorAll, high1, high2, high3, high4;
extern unsigned int totalTicks;
extern unsigned char motor_on;

int setDutyCycle(unsigned int throttle, unsigned int motor_num);
void motors_init(void);
void start_motor_timers(void);
void motors_on(unsigned char on_noff);
unsigned int controlled_speed(void);
unsigned int normalize_speed(double speed);
#endif
```

*motor_drive.c*

```
#include "pwm.h"
#include "motor_drive.h"

unsigned int speed_report_count;
unsigned char motor_on;

void motors_init(void)
{
    // enable timer0,1,3,5 for motor control
    T0CONbits.T08BIT = 0;   // set timer0 in 16-bit mode
    T0CONbits.T0CS = 0;     // set timer0 clk source as instruction cycle
    T0CONbits.PSA  = 0;     // enable timer0 pre-scale
    T0CONbits.T0SE = 0;     // increment on low to high transition
    T0CONbits.T0PS = 0b001; // choose a pre-scale of 4
    // turn on Timer0
    T0CONbits.TMR0ON = 0;

    // enable timer1
    T1CONbits.TMR1CS = 0b00; // CLK src FOSC/4
    T1CONbits.T1CKPS = MOTOR_TIMER_PRESCALE;
    T1CONbits.T1SOSCEN = 0; // disable secondary osc
    T1CONbits.T1RD16 = 1;   // sing 16-bit-read/write operation enable
    T1CONbits.TMR1ON = 0;   // enable tmr1

    // enable timer3
    T3CONbits.TMR3CS = 0b00; // CLK src FOSC/4
    T3CONbits.T3CKPS = MOTOR_TIMER_PRESCALE;
    T3CONbits.T3SOSCEN = 0; // disable secondary osc
    T3CONbits.T3RD16 = 1;   // sing 16-bit-read/write operation enable
    T3CONbits.TMR3ON = 0;   // enable tmr3

    // enable timer5
    T5CONbits.TMR5CS = 0b00; // CLK src FOSC/4
```

```
    T5CONbits.T5CKPS = MOTOR_TIMER_PRESCALE;
    T5CONbits.T5SOSCEN = 0; // disable secondary osc
    T5CONbits.T5RD16 = 1;   // sing 16-bit-read/write operation enable
    T5CONbits.TMR5ON = 1;   // enable tmr5

    motor_on = 0;
    setDutyCycle (6, 1);      // dutyCycle = 10%, frequency = 50Hz
    setDutyCycle (6, 2);      // dutyCycle = 10%, frequency = 50Hz
    setDutyCycle (6, 3);      // dutyCycle = 10%, frequency = 50Hz
    setDutyCycle (6, 4);      // dutyCycle = 10%, frequency = 50Hz

    PORTBbits.RB0 = 1;
    PORTBbits.RB1 = 1;
    PORTBbits.RB2 = 1;
    PORTBbits.RB3 = 1;
    speed_report_count = 1;
}

// 1: start the engines. 0: stop the engines.
void motors_on(unsigned char on_noff)
{
    motor_on = on_noff;
}

// throttle is a number between 0 and 16000
int setDutyCycle(unsigned int throttle, unsigned int motor_num)
{
    if (throttle < 16001 && throttle > 0) {
        if (motor_num == 1 || motor_num == 0){
            high1 = throttle + THROTTLE_LOW;
        }
        if (motor_num == 2 || motor_num == 0){
            high2 = throttle + THROTTLE_LOW;
        }
        if (motor_num == 3 || motor_num == 0){
            high3 = throttle + THROTTLE_LOW;
        }
       if (motor_num == 4 || motor_num == 0){
            high4 = throttle + THROTTLE_LOW;
        }
        return 1;
    }else {
        return 0;
    }
}

void start_motor_timers(void)
{
    unsigned int temp;
    INTCONbits.TMR0IF = 0;
    temp = ~high1;
```

```
    TMR0H = temp >> 8;
    TMR0L = temp;
    T0CONbits.TMR0ON = 1;

    PIR1bits.TMR1IF = 0;
    temp = ~high2;
    TMR1H = temp >> 8;
    TMR1L = temp;
    T1CONbits.TMR1ON = 1;

    PIR2bits.TMR3IF = 0;
    temp = ~high3;
    TMR3H = temp >> 8;
    TMR3L = temp;
    T3CONbits.TMR3ON = 1;

    PIR5bits.TMR5IF = 0;
    temp = ~high4;
    TMR5H = temp >> 8;
    TMR5L = temp;
    T5CONbits.TMR5ON = 1;
}

//translate omega1,2,3,4 to pwm1,2,3,4 and actuate motors accordingly
unsigned int controlled_speed(void)
{
    unsigned int t1;
    unsigned int t2;
    unsigned int t3;
    unsigned int t4;
    if (controller_param->FCS == FCS_MOTOR_TEST) {
        setDutyCycle(test_speed[0], 1);
        setDutyCycle(test_speed[1], 2);
        setDutyCycle(test_speed[2], 3);
        setDutyCycle(test_speed[3], 4);
    } else {
        //actuate the motors
        if (motor_on) {
            t1 = normalize_speed(ctrl_data->omega1square);
            t2 = normalize_speed(ctrl_data->omega2square);
            t3 = normalize_speed(ctrl_data->omega3square);
            t4 = normalize_speed(ctrl_data->omega4square);
            setDutyCycle(normalize_speed(ZERO_SPEED_THROTTLE + ctrl_data-
>omega1square), 1);
            setDutyCycle(normalize_speed(ZERO_SPEED_THROTTLE + ctrl_data-
>omega2square), 2);
            setDutyCycle(normalize_speed(ZERO_SPEED_THROTTLE + ctrl_data-
>omega3square), 3);
            setDutyCycle(normalize_speed(ZERO_SPEED_THROTTLE + ctrl_data-
>omega4square), 4);
            if (speed_report_count == SPEED_REPORT_RATE) {
```

```
            // printf("T1: %u, T2: %u, T3: %u, T4: %u.\n\r", t1, t2, t3, t4);
             //print the motor speeds
            /*
            printf("M1: ");
            printFloat(ctrl_data->omega1square);
            printf(", M2: ");
            printFloat(ctrl_data->omega2square);
            printf(", M3: ");
            printFloat(ctrl_data->omega3square);
            printf(", M4: ");
            printFloat(ctrl_data->omega4square);
            printf(".\n\r");

            printf("e_roll: ");
            printFloat(kinetic_errs->roll_error);
            printf("e_pitch: ");
            printFloat(kinetic_errs->pitch_error);
            printf(", e_yaw: ");
            printFloat(kinetic_errs->yaw_error);
            printf(", e_alt: ");
            printFloat(kinetic_errs->altitude_error);
            printf(".\n\r");
            printf(".\n\r");
*/
            speed_report_count = 0;
         } else {
            speed_report_count++;
         }

      } else {
         // gives zero throttle to all
         setDutyCycle(6, 0);
      }
   }
}

// normalize omega2 into throttle input
unsigned int normalize_speed(double speed) {
   int output;
   output = (int)SPEED2THROTTLE_FACTOR*speed;
   output = (output < 0) ? 0:
           (output > MAX_THROTTLE) ?  MAX_THROTTLE :
            output;
   return (unsigned int)output;
}
/*
void pwm_init(void)
{
   SetOutputEPWM1(SINGLE_OUT, PWM_MODE_1);
   OpenEPWM1(0xFF, ECCP_1_SEL_TMR12);
```

```
    SetDCEPWM1(0x6FF);
}
*/
```