THE UNIVERSITY OF TEXAS – PAN AMERICAN

ELECTRICAL ENGINEERING DEPARTMENT

QUAD-S

Quadcopter Unmanned Autonomous Delivery System

Final Report

ELEE 4361.01

Fall 2012

Wednesday, December 5, 2012

Senior Designers:

Edson Gallegos

Alex Marroquin

Angel Tijerina

Primary advisor:	Dr. Weidong Kuang
Secondary advisor:	Dr. Yul Chu
Instructor:	Dr. Edward Banatoski

1 Table of Contents

2 List of Figures

3 List of Tables

4 Abstract

We design and implement three critical components for an autonomous quadcopter capable of delivering packages from a source to a destination. We focus on the Stability System, Control System, and Infrared Tracking and Communication. We test each system and then evaluate their performance to determine the final system integration.

5 Introduction

During an emergency or time of war, the ease of supply transportation to locations in need is critical to mission success. However, the local conditions may not allow for easy transport on land. Air support is another option, but may not be possible if all forces are already employed with other tasks in the mission. In addition, manned air support may also put more human lives at risk. A mode of transportation that is both unmanned and requires minimal human input is necessary to minimize risk while still providing essential supplies to those who need them. The QUAD-S provides a solution to unmanned supply delivery.

The QUAD-S will have infrared sensors to detect the position of packages and targets (destinations) marked with IR transmitters. The IR transmitters will broadcast a signal, which will identify them as a package or target. Within the same signal will be an ID number for when there are multiple packages going to different locations. The QUAD-S will always deliver the current package it is holding before searching for, and picking up, other packages and dropping them off at their destinations. Each package and target will come in pairs i.e. a certain package has to be delivered to a certain target. The QUAD-S will continue this process of searching for a retrieval package, taking it to its destination, then finding the closest package and taking it to its destination. Once the QUAD-S no longer detects a retrieval signal or after a set time, it will return to a starting location or home base.

The QUAD-S will be useful in situation where GPS is inaccessible but supplies need to be delivered to an isolated location. The people in need of supplies may be on the move to avoid hazards and conflict. If they are carrying an emergency infrared transmitter, the quadcopter will be able to track them and leave supplies at their location. The unmanned nature of the QUAD-S also means it can be used in other hazardous situations such as the movement of dangerous chemicals. After a natural disaster or during construction the quadcopter can be used to carry supplies or rubble that has been marked and take it to the construction area or dumpsite. With more advanced systems, a large-scale QUAD-S could also act as a search and rescue vehicle, finding marked people in need of medical assistance and taking them to hospitals or other safe areas.

6 Design

The overall QUAD-S system is shown below in the block diagram:

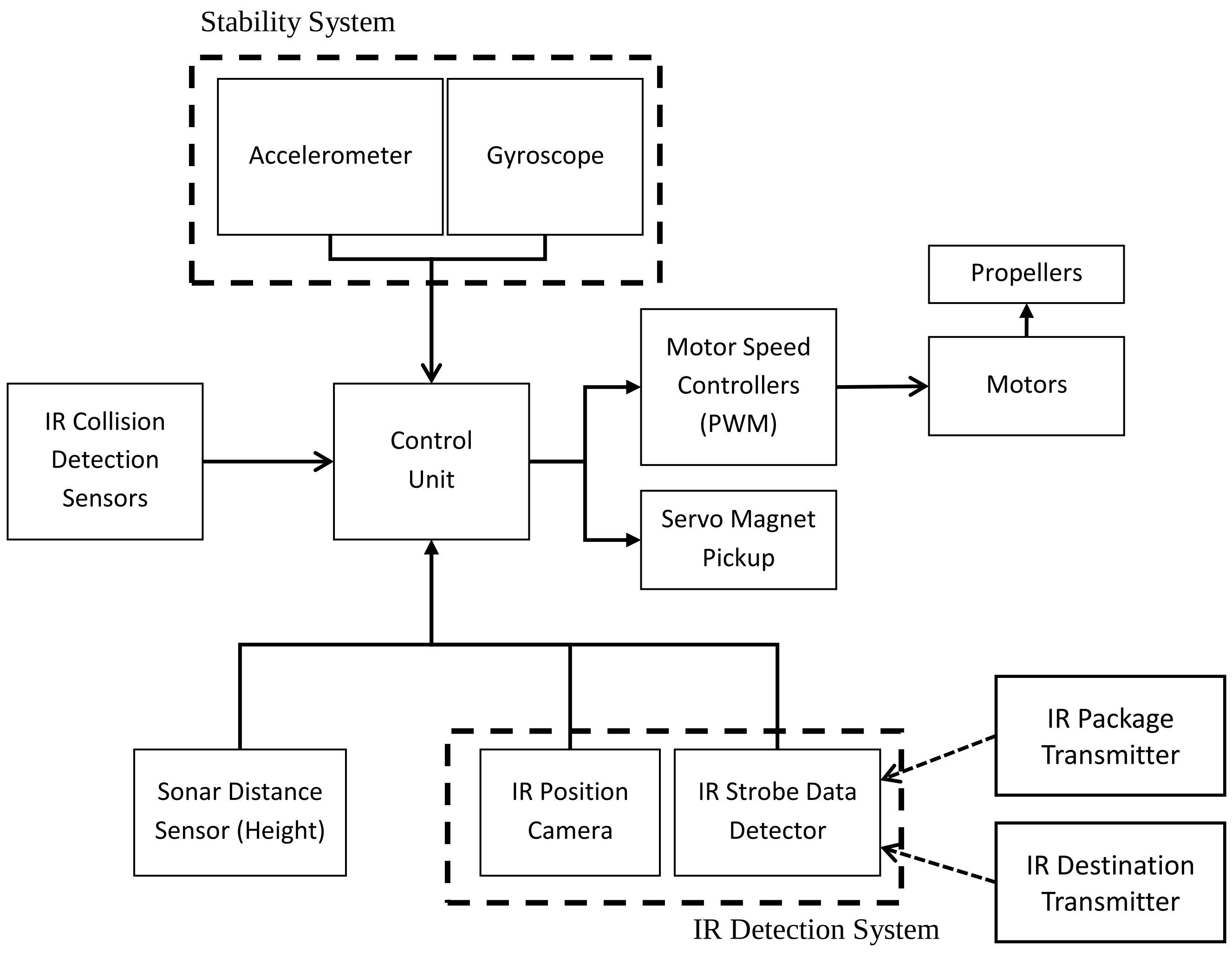


Figure 1: Block diagram for QUAD-S

The overall system is simple and self-contained due to its autonomous nature. The four sensing systems: stability, IR Detection, Sonar, and IR Collision tell the controller how the quadcopter should move at any point in time. The controller then sends the signals to the motor speed controllers, adjusting the motor speed to match the motion that is required, such as stabilization and moving toward a target. Finally, the controller also sends a signal to the magnetic arm servo to lower the arm when picking up a package and raise the arm when dropping it off at the target.

The QUAD-S is divided into the following subsystems:

- Control Unit
- Stability System
- IR Detection System
- IR Package/Destination Transmitters
- IR Collision Detection Sensors
- Sonar Height Sensor
- Motor Speed Controllers
- Motors/Propellers
- Servo Magnet Pickup

6.1 Stability System

In order to control the quadcopter in all aspects since its take off onto its hover and flight motions and finally onto its landing we had to work on a stability system, which would help us by measuring distinct characteristics of the motion of the quadcopter. This data would then be feedback to the control system in order to create the necessary adjustments for the quadcopter to be as stable as possible and perform the desired pickups and deliveries, the control system will be explained in detail later in this report.

One of the most important components for our stability system was an Inertial Measurement Unit (IMU) that would allow us to measure and report the orientation of the quadcopter. The sensor we ended up implementing was the Invensense MPU6050 [1], which is a combined 3-axis gyroscope and 3-axis accelerometer into one chip. This sensor was chosen because it would allow us to measure the orientation and the acceleration of the quadcopter for stability purposes and because this sensor could be interfaced via I^2C communication with a wide range of microcontrollers including Microchip PICs and Arduino. In addition, this chip came in a breakout board, which would get rid of any axes misalignments that would have resulted from trying to stack an accelerometer and a gyroscope separately.

The decision behind selecting I^2C communication was because it is commonly used in applications like this one where there is a need to communicate some sensors with the main microcontroller and also because of the simplicity of the communication itself as this one only needs two lines to complete the communication, a data line and a clock line [2]. We used the Register Map from Invensense on the MPU-6050 to determine from which registers we needed to read data [3].

From the 3-axis gyroscope of our sensor we were able to measure the orientation (Euler angles: Yaw, Pitch, and Roll) of our system. A visualization of the motion of these angles acting on a quadcopter in the xyz plane can be seen in Figure 2.

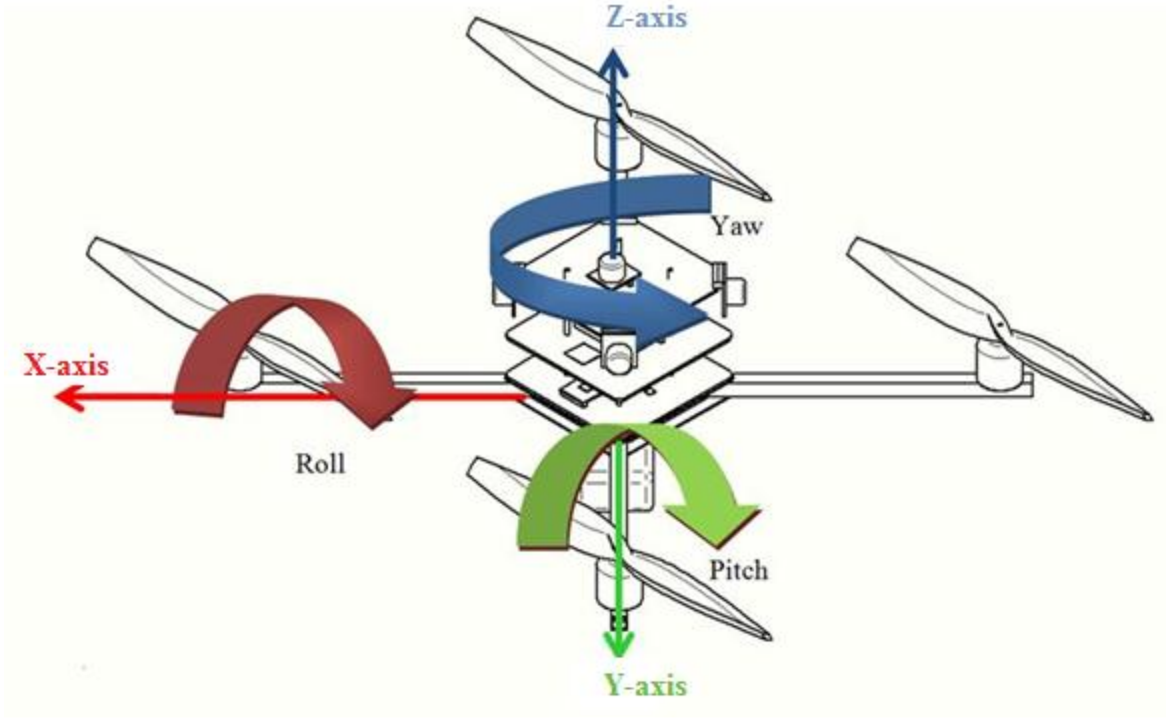


Figure 2: Euler angles in the XYZ plane

After encountering some difficulties, mostly inefficient coding when trying to interface the sensor with the Microchip PIC 16F884 microcontroller, we decided to utilize an ATMEGA168-20PU microcontroller available on the Arduino-Uno board. The reason behind the switch was because in our initial research we found a lot more information and coding for similar projects with this type of microcontroller and it could also be interfaced using I^2C.

We used a program that we came across in our initial research by Jeff Rowberg [4] that would get the data from our sensor and do all the necessary calculation and conversions from the output voltage the sensor outputs to the Euler angles we then needed for the control system. We uploaded the code to the ATMEGA168-20PU microcontroller and connected the Arduino-Uno board with the sensor as shown in Figure 3, a picture of the actual devices connected is shown in Figure 4. We connected the SDA and SCL I^2C lines with each other for both devices, and wired the sensor based on the schematic given on the seller's website under "Documents" and then schematic [5] this schematic is shown in Figure 5. Here we just added an exception, the VLOGIC pin was connected to the VCC pin on the breakout board of the sensor. In addition, Figure 4 how the Arduino-Uno board allowed us to power both devices without the use of a power supply as it has an integrated power supply of 3.3V and 5V, we used the 3.3V and this made the wiring relatively simpler.

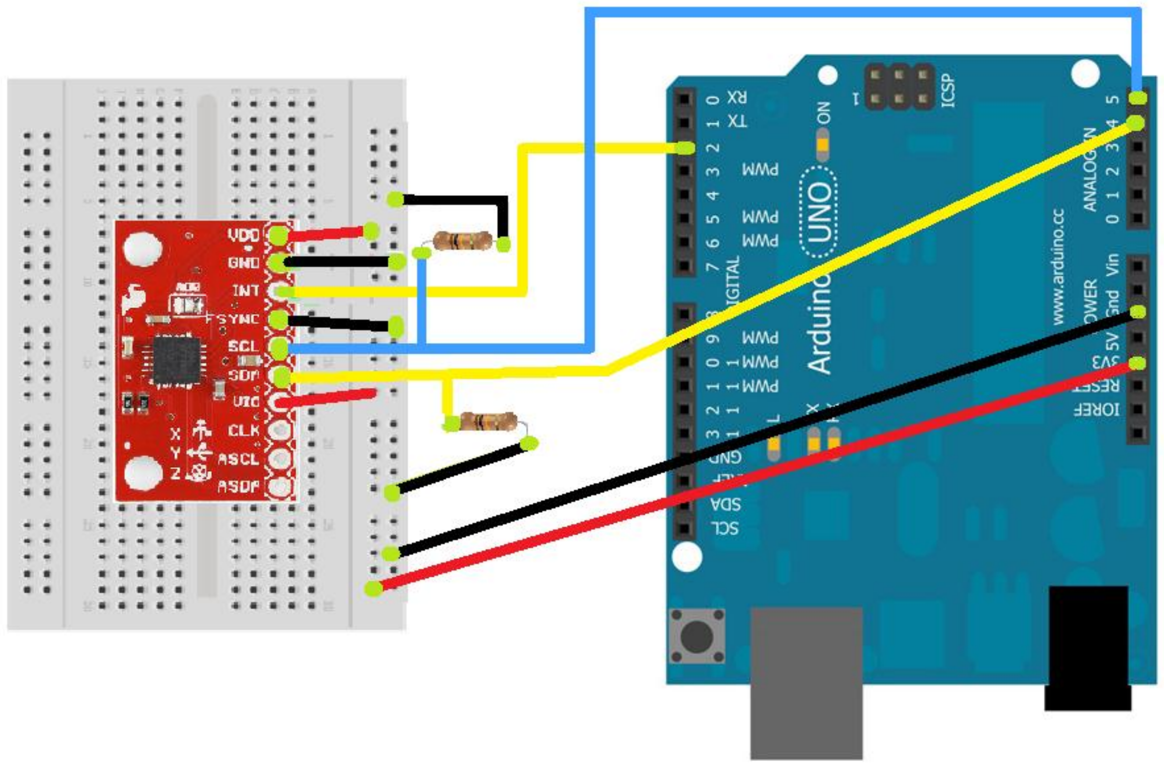


Figure 3: Acc/Gyro Sensor Schematic

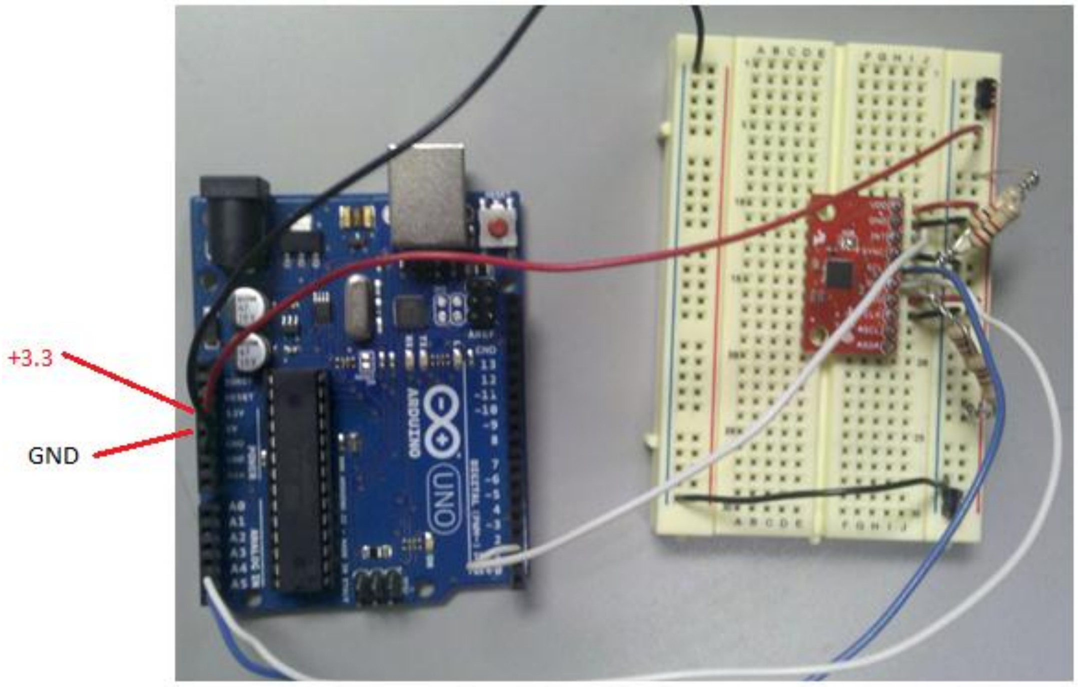
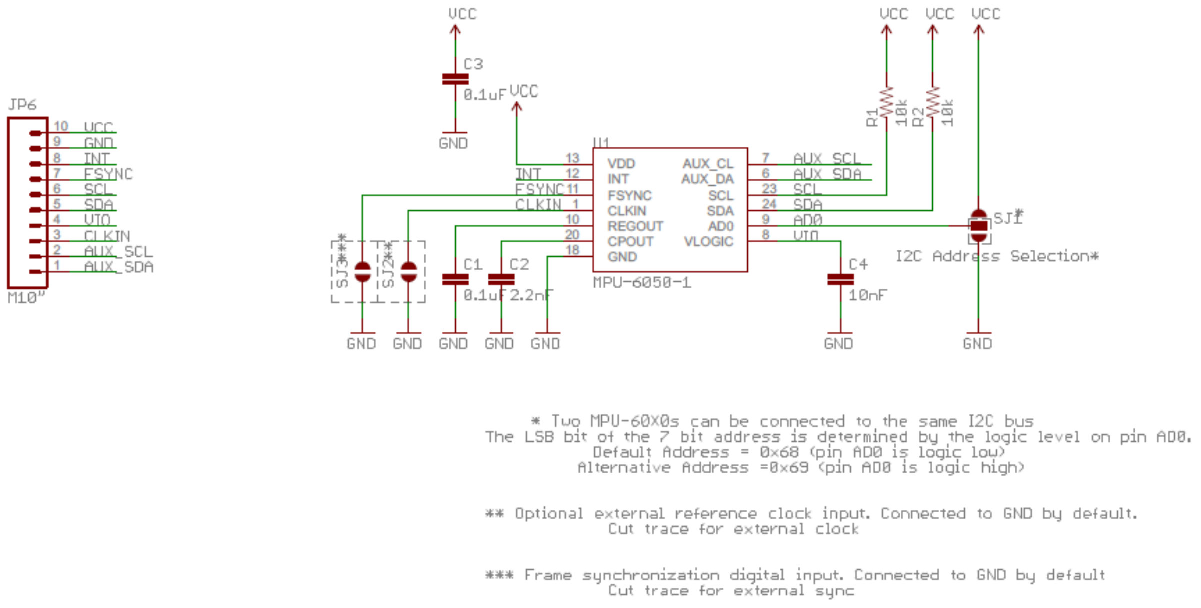


Figure 4: Actual Acc/Gyro and Arduino Circuit



* Two MPU-60X0s can be connected to the same I2C bus
The LSB bit of the 7 bit address is determined by the logic level on pin AD0.
Default Address = 0x68 (pin AD0 is logic low)
Alternative Address =0x69 (pin AD0 is logic high)

** Optional external reference clock input. Connected to GND by default.
Cut trace for external clock

*** Frame synchronization digital input. Connected to GND by default
Cut trace for external sync

Figure 5: Wiring of the MPU6050 sensor breakout board

When attaching the sensor to the breadboard we tried to level it so that it was at the center of the breadboard so that when we took our measurements the movement of the breadboard would be leveled with the breakout board of the sensor. Figure 6 shows the reference orientation and polarity for the sensor that helped us determine and read our data values, this one was taken based on the datasheet of the sensor [1].

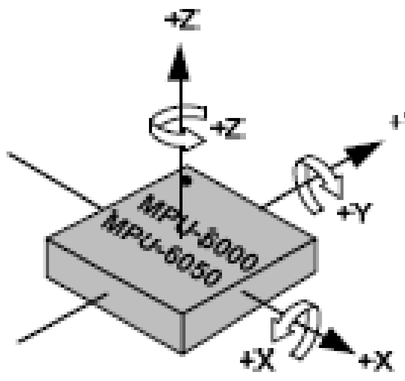


Figure 6: MPU-6050 Axis Reference

6.2 Control Systems

6.2.1 Control System Required

In order to allow the quadcopter to move or hover, its orientation has to be controlled. The quadcopter orientation depends on the angles about the x, y, and z-axes. These angles are the yaw, pitch, and roll. The values of the angles are controlled by the motors thus the control system has to be able to calculate the appropriate motor voltages. To calculate the best values we decided to use a PID control. The PID calculates the control signal by taking into account the desired value, current value, accumulated error, and the rate of change of error. Each component of the PID is explained below [6].

6.2.2 Proportional Control (P)

This term outputs the change that should occur for the actual value to reach the desired value. It only depends on and is directly proportional to the current error. This term is the main driver in reducing the error of the system since it produces the largest visible changes. Large proportional terms cause the actual values to change by huge margins which can cause instability. This is remedied by using a smaller proportional term. Proportional control tends to cause a steady state error, in other words the actual value settles at a value different from the desired value. This can be fixed by adding integral control. The proportional term is defined by the following equation.

$$P = K_P(desired - actual)$$

6.2.3 Integral Control (I)

The integral part takes into account the accumulated error or history of the system by getting the sum of the current error over time i.e. the difference between accumulated areas under the curves made by the actual values and the desired values. It then gives a control signal for the error that should have been corrected previously. This control is good for reducing small error, which reduces the steady state error. The drawback of this term is that it tends to temporarily overshoot the desired value. This can be fixed by either reducing the integral term or by providing damping through the derivative control. The equation for the integral term is shown below.

$$I = K_I \int e(t)dt\,, \qquad e(t)\text{: error, given by } e(t) = desired - actual$$

6.2.4 Derivative Control (D)

The derivative control affects the slope, or rate of change, of the error over time by predicting what the future value may be from the slope. This dampens the response of the proportional and integral controls by slowing down the change in the control signal. It is useful for reducing overshoot but it also causes the overall response to be slower. This can be fixed by increasing the proportional or integral control. The derivative control is also highly sensitive to noise, which can cause it to produce chatter or rapid oscillations in the signal. This chatter can damage the motor so it must be filtered away or completely avoided. Derivative control is given by the following equation.

$$D = K_D \frac{de(t)}{dt}, \qquad e(t)\text{: error}$$

The control signal is calculated by a function of the three terms. To achieve the best response the gain of each term has to be tuned. This requires knowledge of the system variables and constants. For this semester, we assumed the values of all the constants and only considered the system variables since the constants can be easily replaced later on.

6.2.5 PID vs. PID and Kalman Filter

We worked on the overall design of the control system. At first, we were only going to use PID control but in the quadcopter project on the Starlino website [7], a PID with a Kalman Filter was used as shown in the figure below [8].

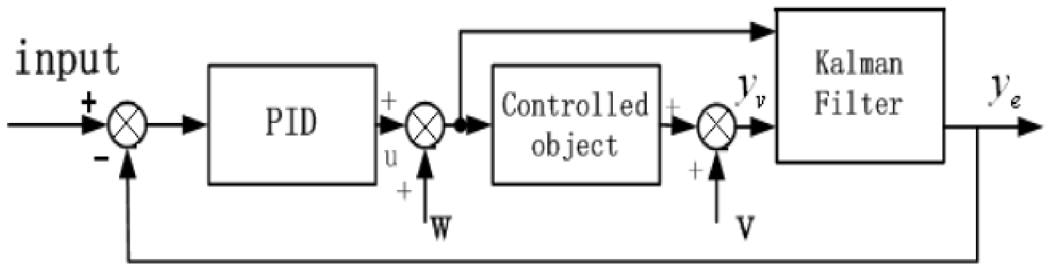


Figure 7: PID with Kalman Filter

The Kalman filter reduces the influence of noise w and v on the calculations, i.e. the controller is less sensitive to disturbances and can control the system more efficiently [9]. Basically, the Kalman filter takes noisy measurements observed over time and outputs an estimate of the unknown variables that is generally more precise than those based on a single measurement [10]. The Kalman filter can be implemented using the following equation.

$$X_k = KZ_k + (1 - K)X_{k-1}$$

where

X_k: *Current estimation,* K: *Kalman gain*

X_{k-1}: *Previous estimation,* Z_k: *Measured Value*

This equation can be broken down into an iterative process as shown below.

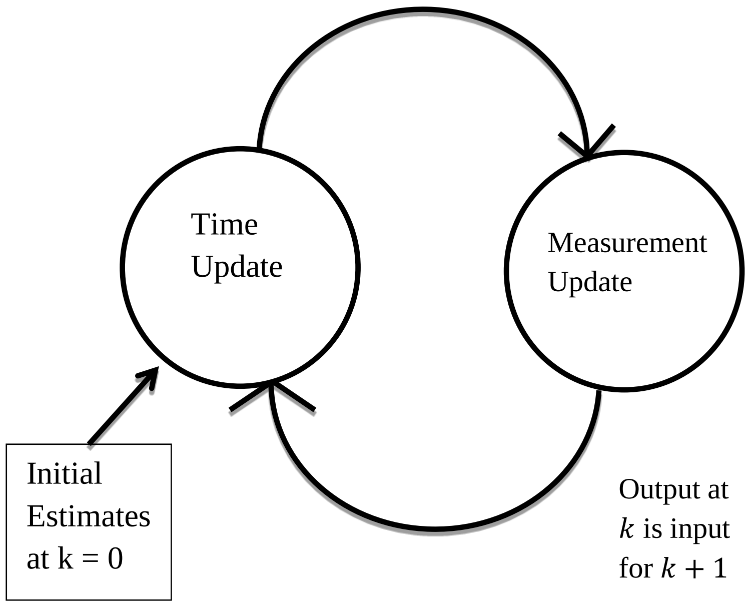


Figure 8: Kalman Filter Update Process

Each update performs a different set of calculations for estimating the current state. The estimation process can be broken down into the following steps [10] [11] [12][13].

$$X_k^- = AX_{k-1} + Bu_k$$

$$P_k^- = AP_{k-1}A^T + Q$$

Measurement Update:

$$K = P_k^- H^T (HP_k^- H^T + R)^{-1}$$

$$X_k = KZ_k + (1 - K)X_{k-1}$$

$$P_k = (1 - KH)P_k^-$$

where

$$X_k^-: Prior\ Estimate, \quad P_k^-: Prior\ Error\ Covariance, \quad Z_k: Measured\ Value$$

$$X_k: Current\ Estimation, \quad P_k: Current\ Covariance, \quad u_k: Control\ Vector$$

$$X_{k-1}: Previous\ Estimated\ State, \quad P_{k-1}: Previous\ Estimated\ Covariance$$

$$K: Kalman\ Gain, \quad Q: Process\ Error, \quad R: Measurement\ Error$$

$$A{:}\,State\ Transition\ Matrix,\quad B{:}\,Control\ Matrix,\quad H{:}\,Observation\ Matrix$$

The prior values refer to the estimates made before iterating once through the process, not to the values calculated in the previous iteration. As seen in [12], the Kalman Filter can produce reliable results but the drawback is figuring out the matrix values, particularly the state transition and observation matrices. To figure out the matrices one needs a detailed system model [14] that can be hard to obtain. This can be circumvented by using a simplified version called the alpha beta filter. The following equation represents the alpha beta filter.

$$X_k = X_{k-1} + \Delta T v_{k-1}$$

where

$$\Delta T{:}\,Sample\ time\ interval,\quad v_{k-1}{:}\,Rate\ of\ change\ of\ system$$

For the alpha beta filter, the rate of change is assumed to remain constant throughout the entire time the system is active as shown below.

$$v_k = v_{k-1}$$

Afterwards the residual, or error, of the system is calculated with the following equation.

$$R_k = Z_k - X_k$$

This residual can then be used to redefine the state estimation and velocity equations as follows.

$$X_k = X_{k-1} + \alpha R_k$$

$$v_k = v_{k-1} + \frac{\beta}{\Delta T} R_k$$

Where α and β are parameters selected to optimize the output. These parameters follow the conditions below.

$$0 < \alpha < 1,\quad 0 < \beta < 2,\quad 0 < 4 - 2\alpha - \beta$$

The algorithm for the alpha beta filter uses fewer equations than the one for the Kalman filter. Additionally, the alpha beta filter requires less parameters dependent on the system yet it can produce similar results [14]. With this in mind, we decided to model the system in SIMULINK by implementing it on a motor control. Since we did not find any alpha beta filter block in SIMULINK, we decided to simulate the Kalman filter. The base case was using only a PID while the test case was with the PID and Kalman Filter. The first four plots show the signals for the PID input, PID output, system output, and all three signals respectively while the last four plots show the signals for the PID with the Kalman Filter.

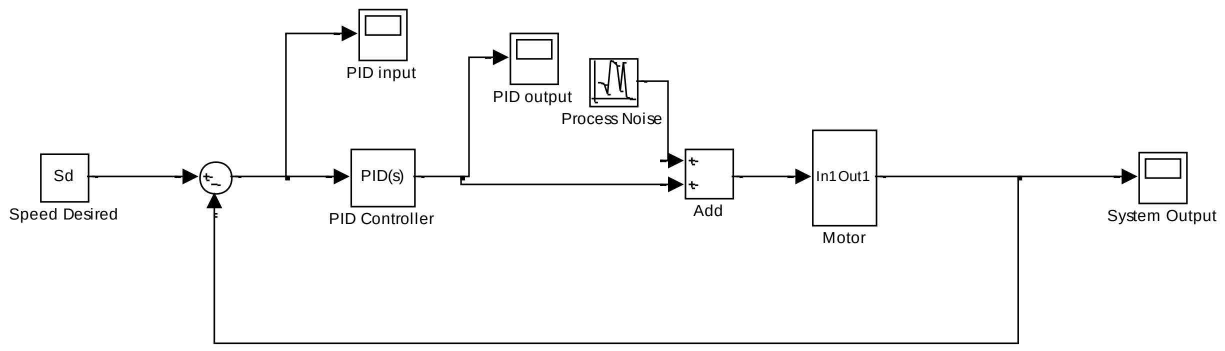


Figure 9: PID only System Model

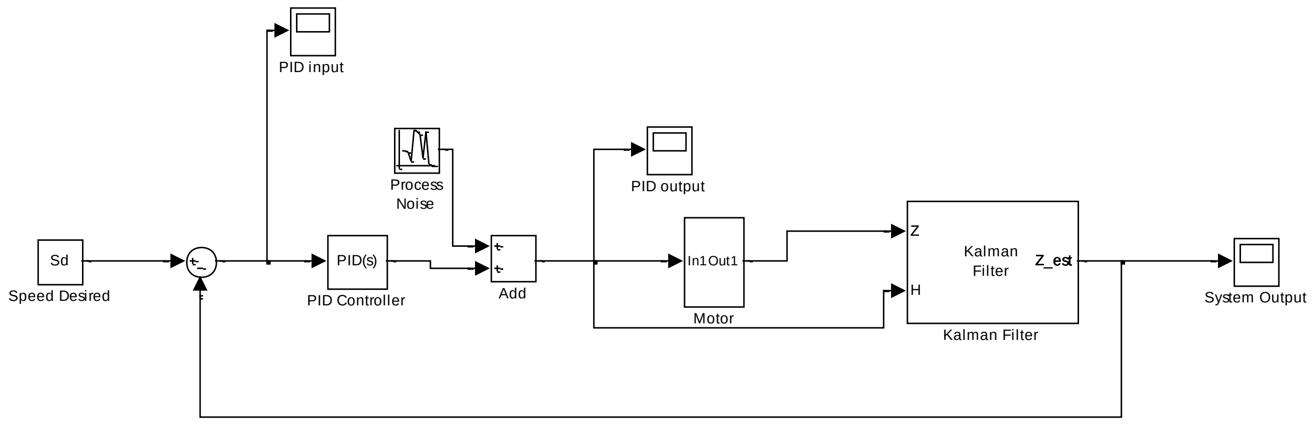


Figure 10: PID with Kalman Filter System Model

PID by Itself

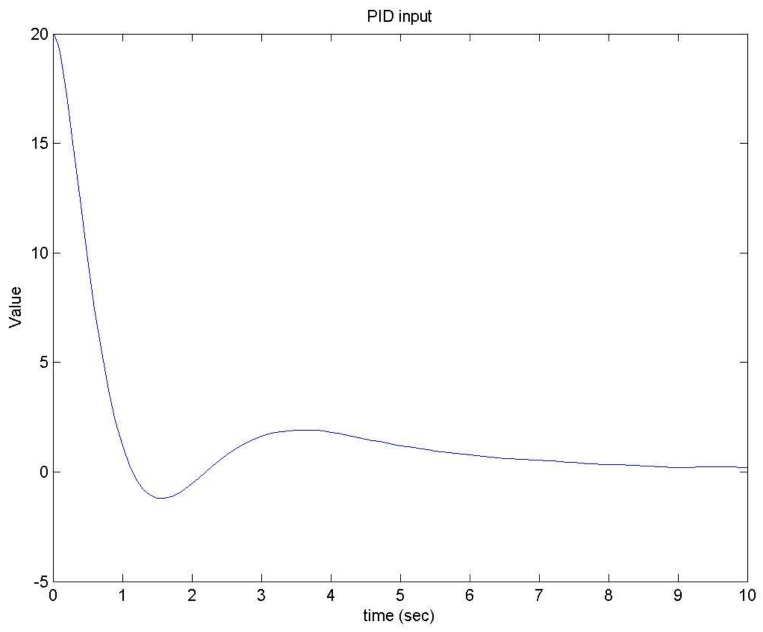


Figure 11: PID input

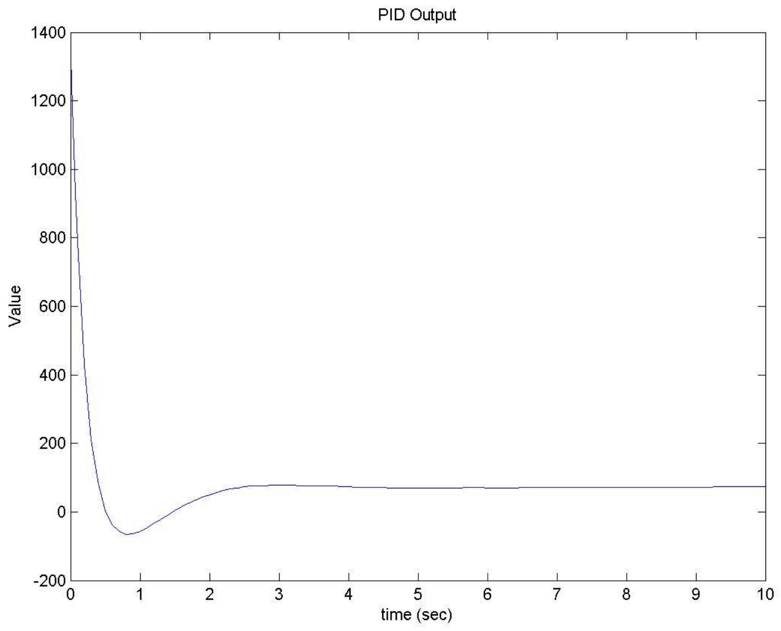


Figure 12: PID output

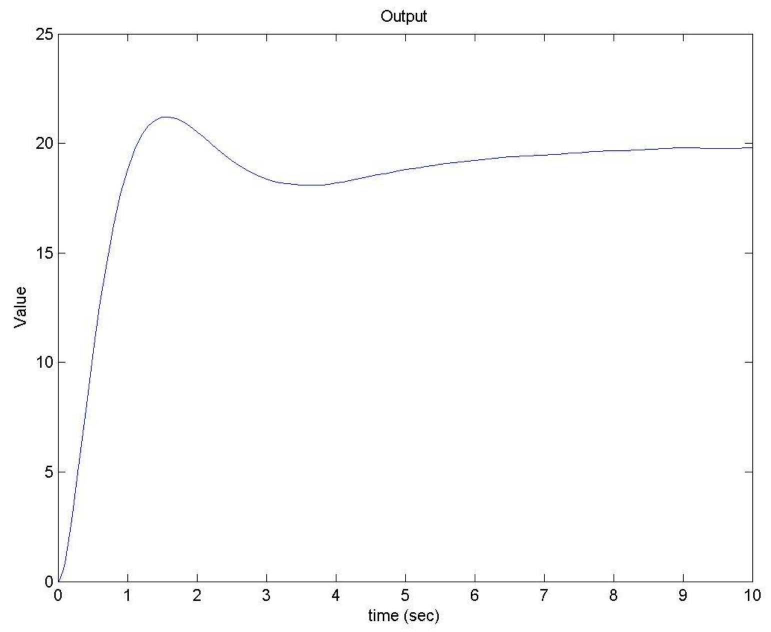


Figure 13: System Output

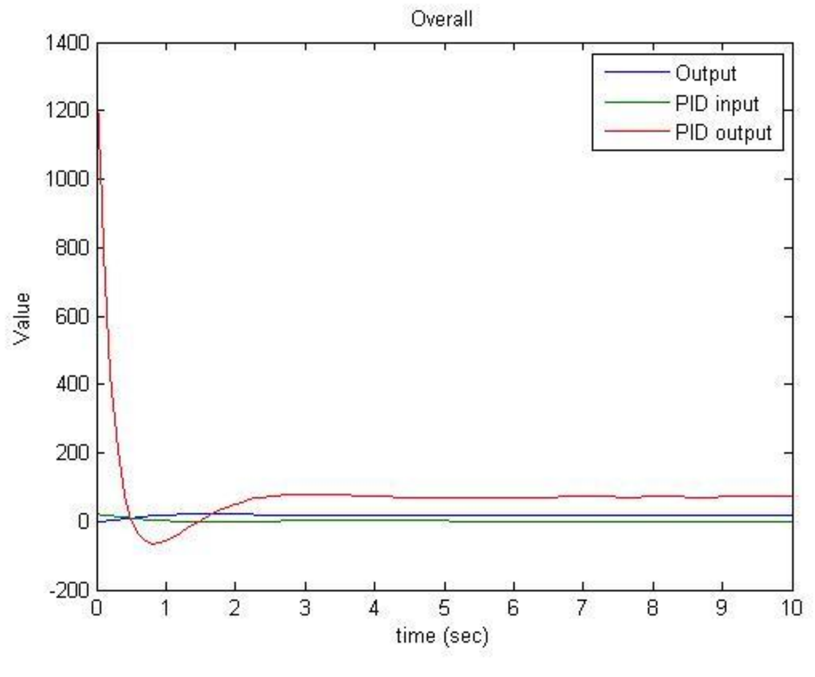


Figure 14: PID input, output, and System output

PID with Kalman Filter

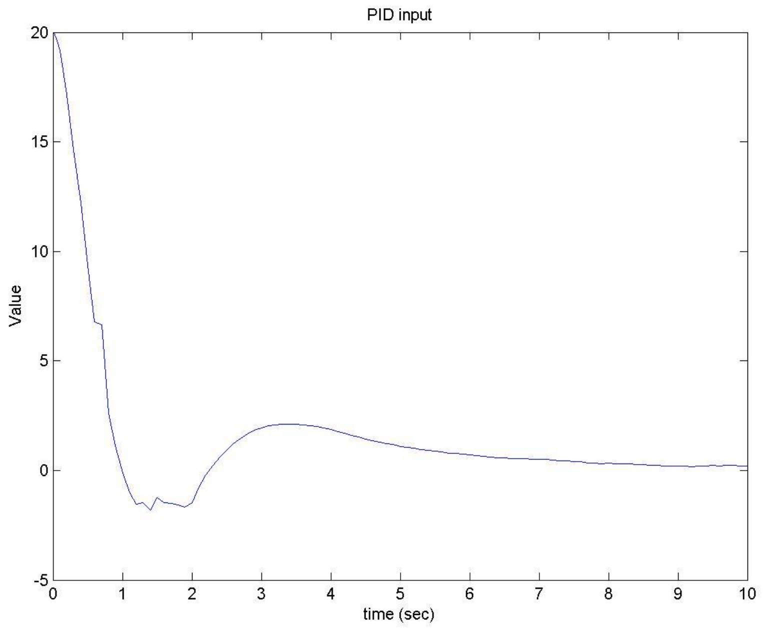


Figure 15: PID with Kalman input

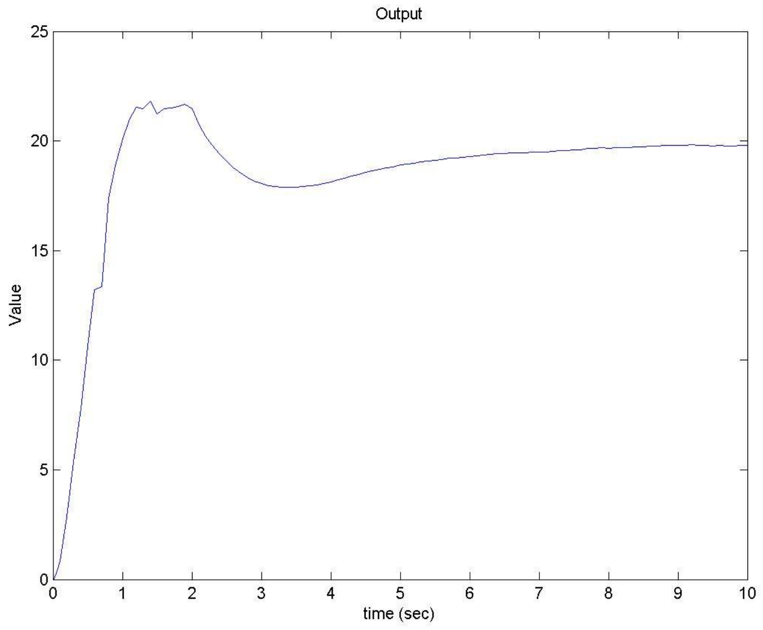


Figure 16: PID with Kalman output

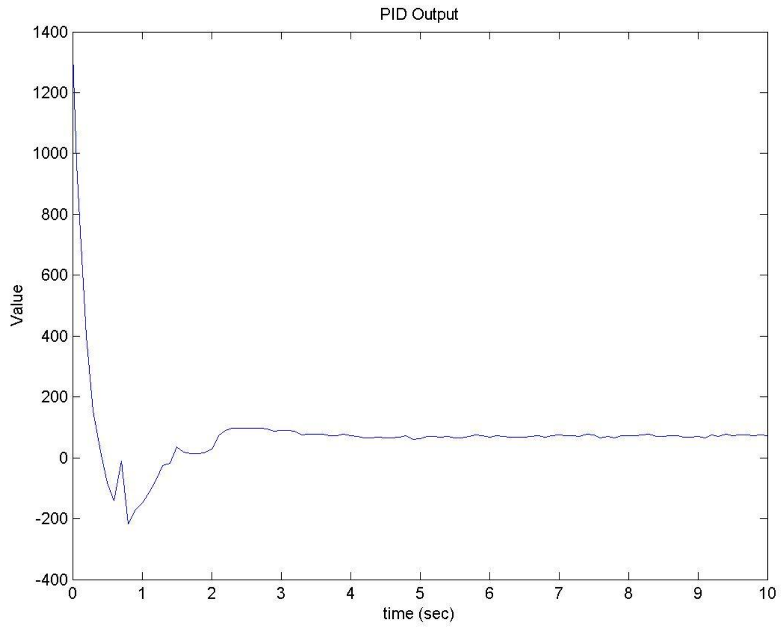


Figure 17: PID with Kalman System output

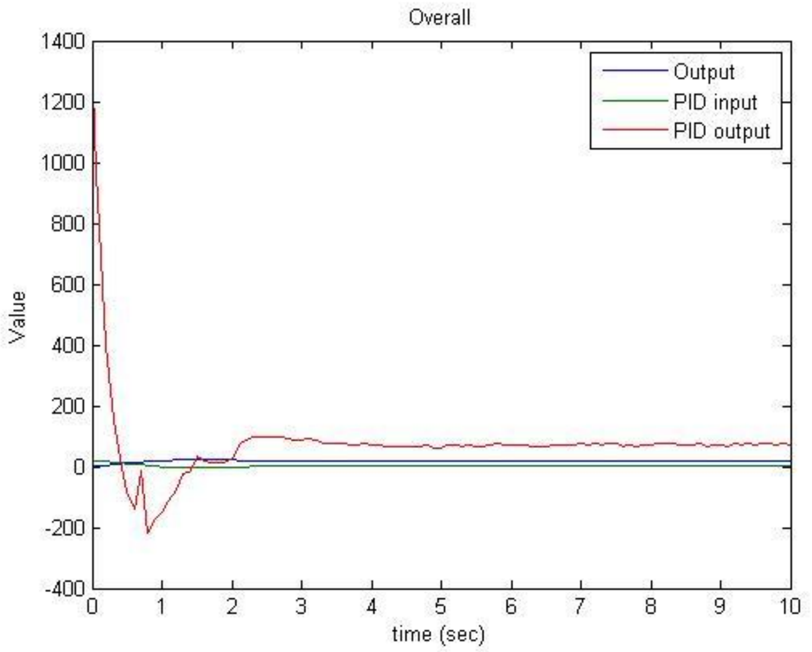


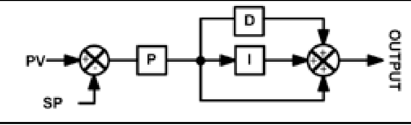
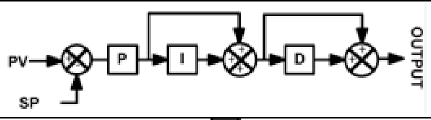
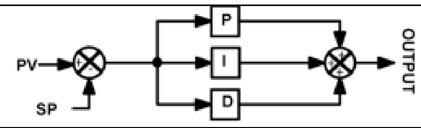
Figure 18: PID with Kalman input, output, and System output

From the simulations, the only PID system had less overshoot and noise than with the Kalman but there were some complications with implementing the Kalman Filter simulations. Despite this, we will first implement the PID only control. If that is not sufficient then we will implement the PID with the Alpha-Beta filter, a simplified version of the Kalman Filter [14].

6.2.6 Different PID Implementations

Before implementing the PID control, we looked at the various types of PIDs. There are 3 types: ideal, parallel, and series, as shown in the figure below [15][16].

Table 1: PID Types

	Laplace Notation	Classical Notation	Block Diagram
Ideal	Output = $Kp\left(1+\frac{1}{Ti(s)}+Td(s)\right)$	$Output = K_p\left[e(t)+\frac{1}{K_i}\int e(t)dt+K_d\frac{de(t)}{dt}\right]$	
Series	Output = $Kp\frac{\left(1+\frac{1}{Ti(s)}\right)\left(1+Td(s)\right)}{1+\frac{Td(s)}{Kd}}$	$Output = K_p\left[e(t)+\frac{1}{K_i}\int e(t)dt\right]\left[1+K_d\frac{de(t)}{dt}\right]$	
Parallel	Output = $Kp+\frac{1}{Ti(s)}+Td(s)$		

Although all of them work well as a PID control, the coefficient tuning is different for each one. Additionally, the output and control signals differ slightly thus creating different intermediate values but the same initial and final values. We decided to test each type in SIMULINK to observe which would be a better control. In our case, both the time required to stabilize and the overshoot are critical to overall stability but overshoot is more important since it has more influence over the entire system. The SIMULINK models for each control are shown below. All the PID models were simulated with the same gains with a target value of 20.

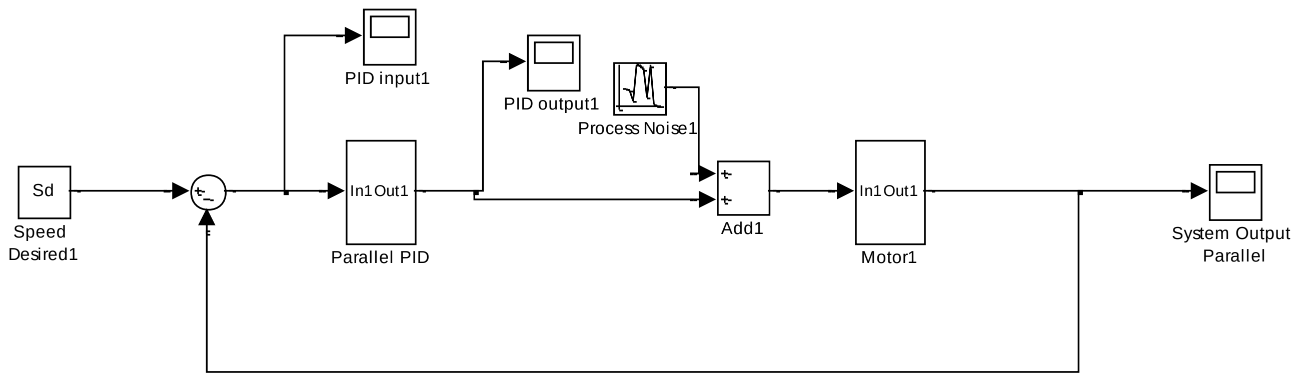


Figure 19: Parallel PID System Model

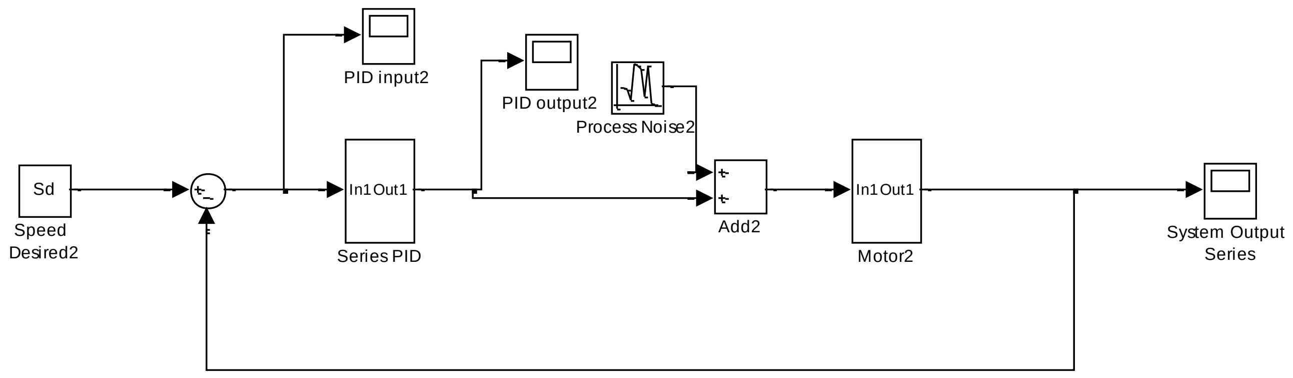


Figure 20: Series PID System Model

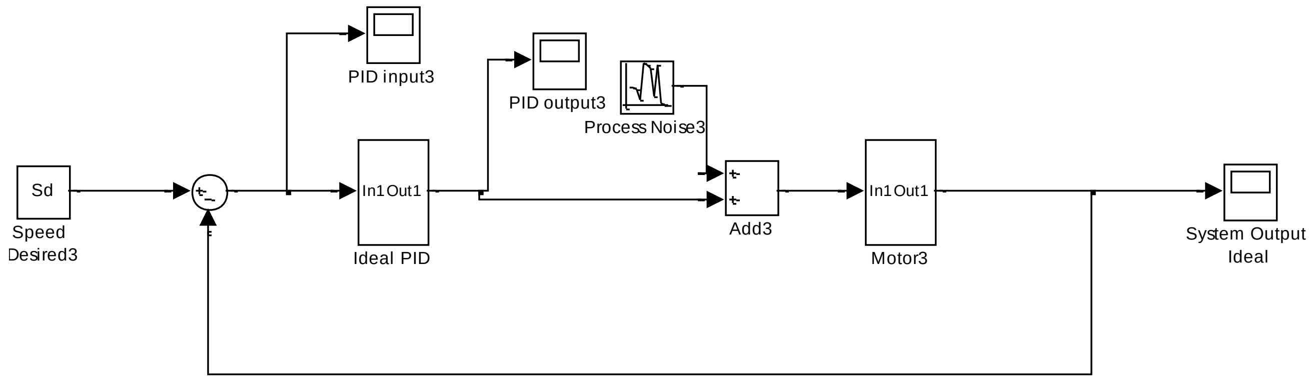


Figure 21: Ideal PID System Model

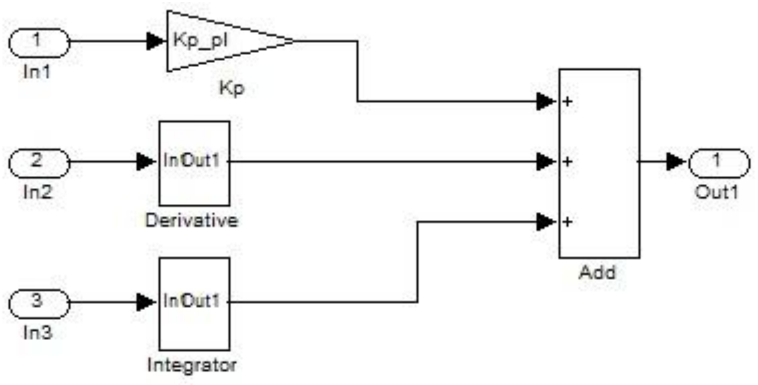


Figure 22: Parallel PID implementation

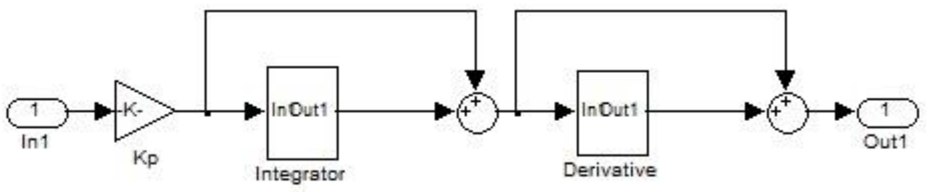


Figure 23: Series PID implementation

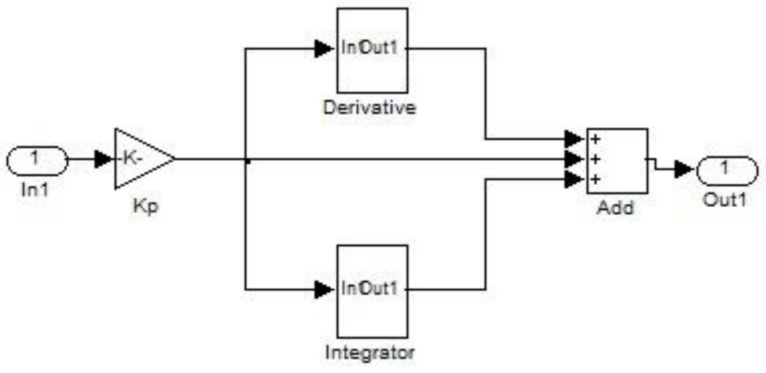


Figure 24: Ideal PID implementation

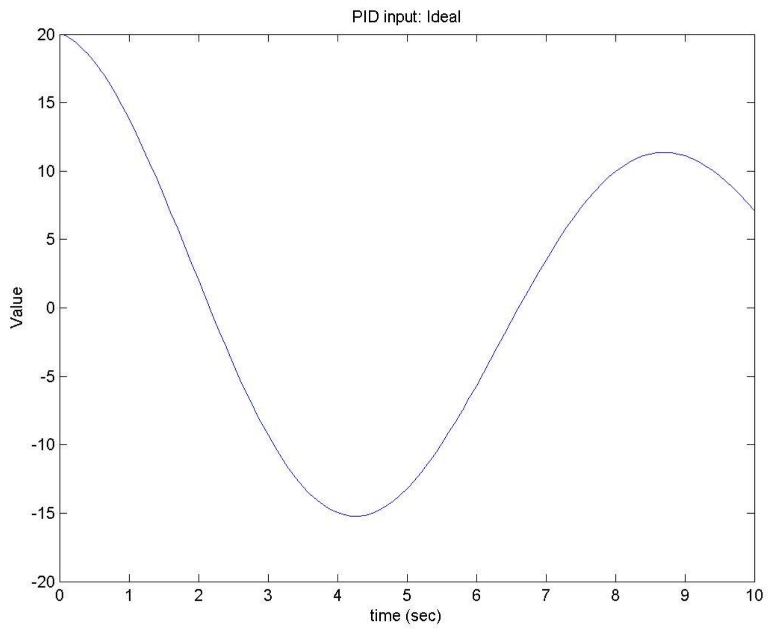


Figure 25: PID input, Ideal

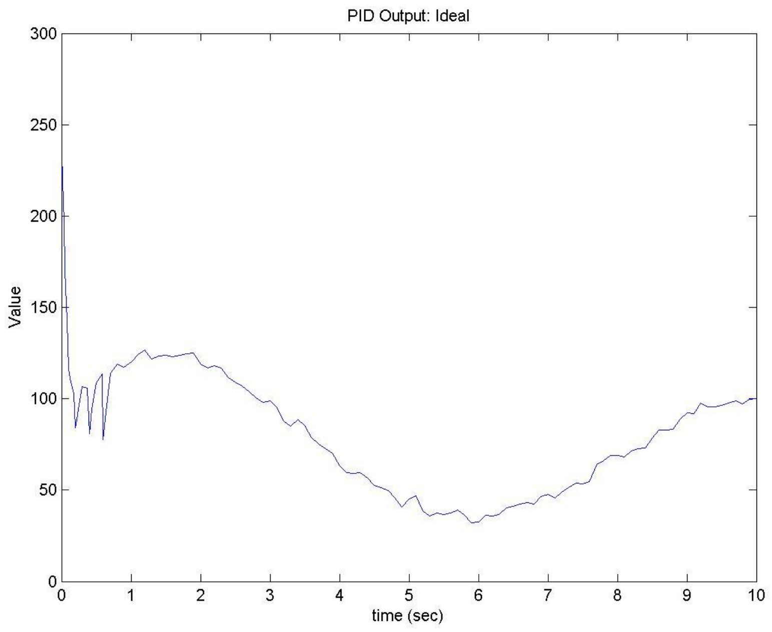


Figure 26: PID output, Ideal

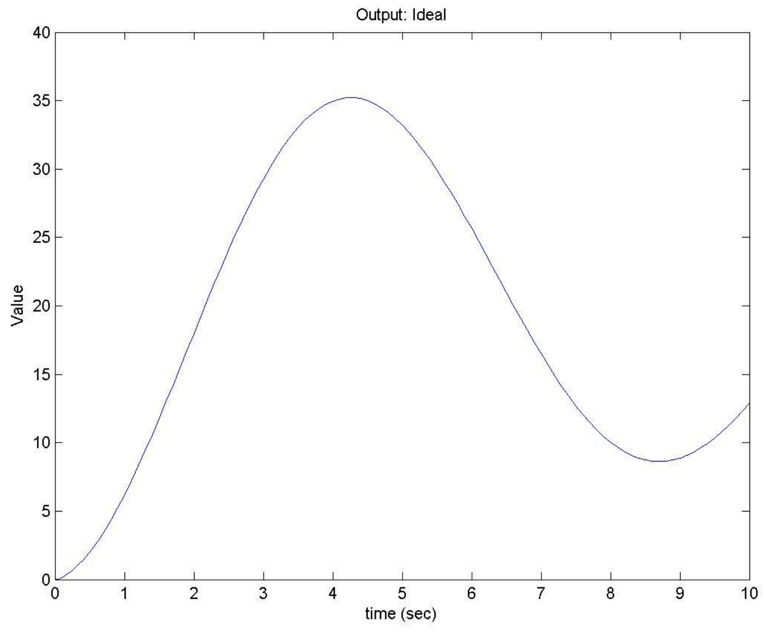


Figure 27: System output, Ideal

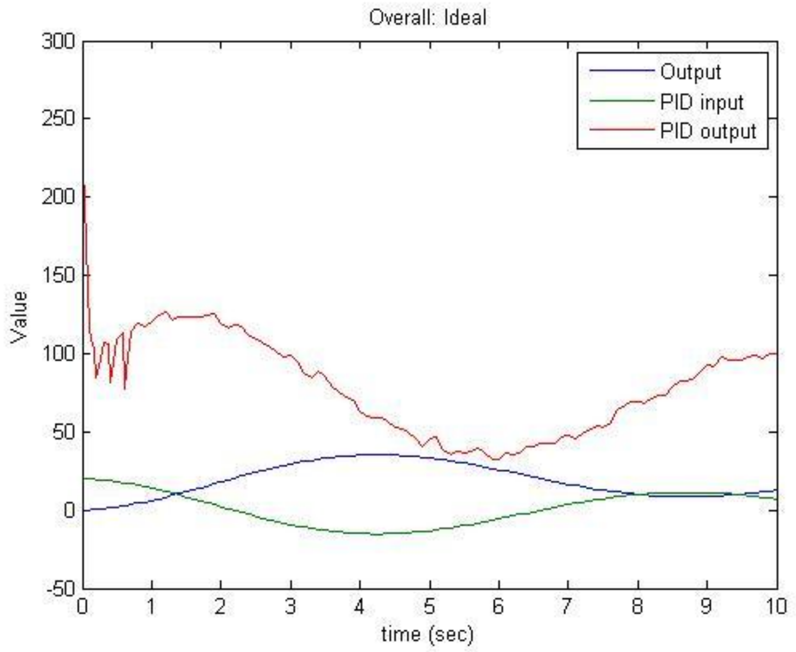


Figure 28: Overall, Ideal

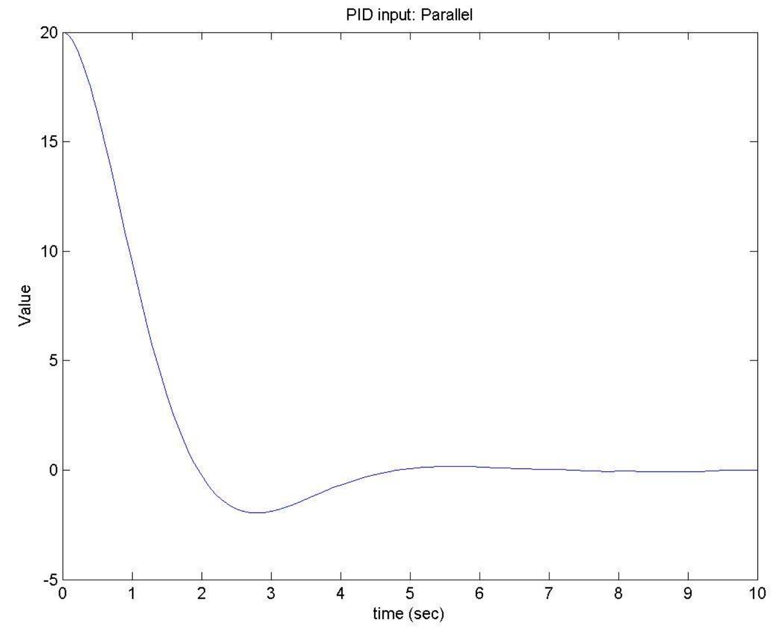


Figure 29: PID input, Parallel

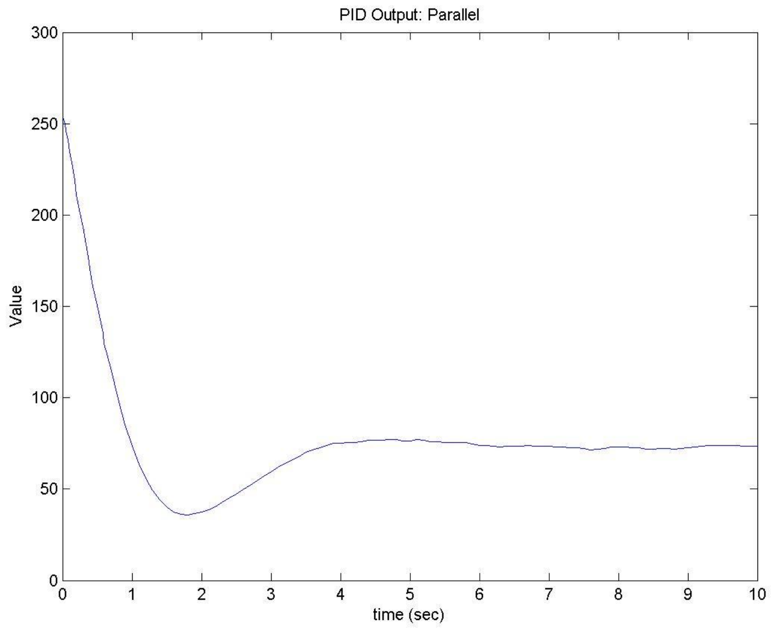


Figure 30: PID output, Parallel

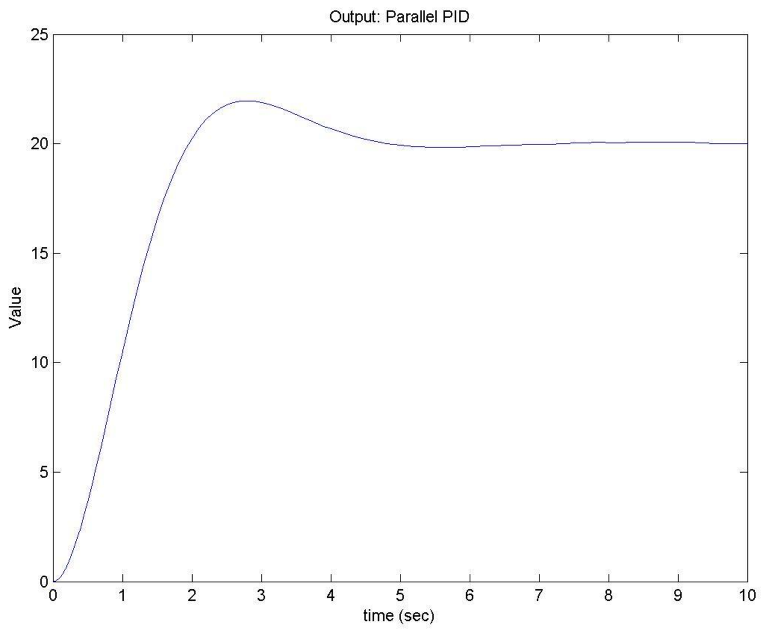


Figure 31: System output, Parallel

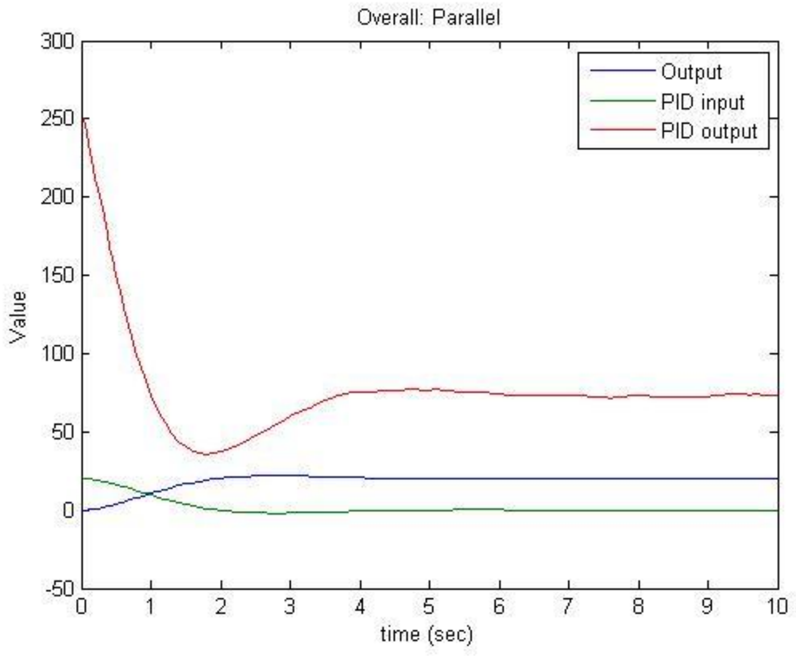


Figure 32: Overall, Parallel

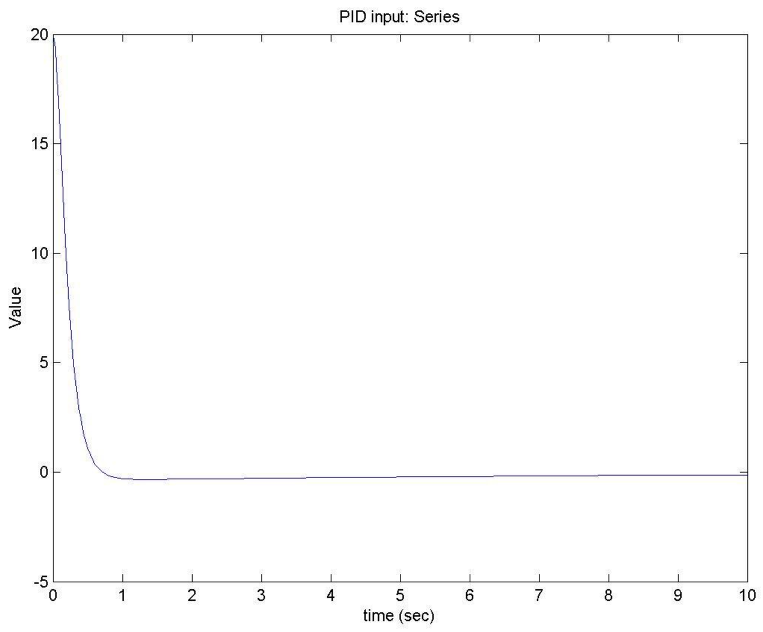


Figure 33: PID input, Series

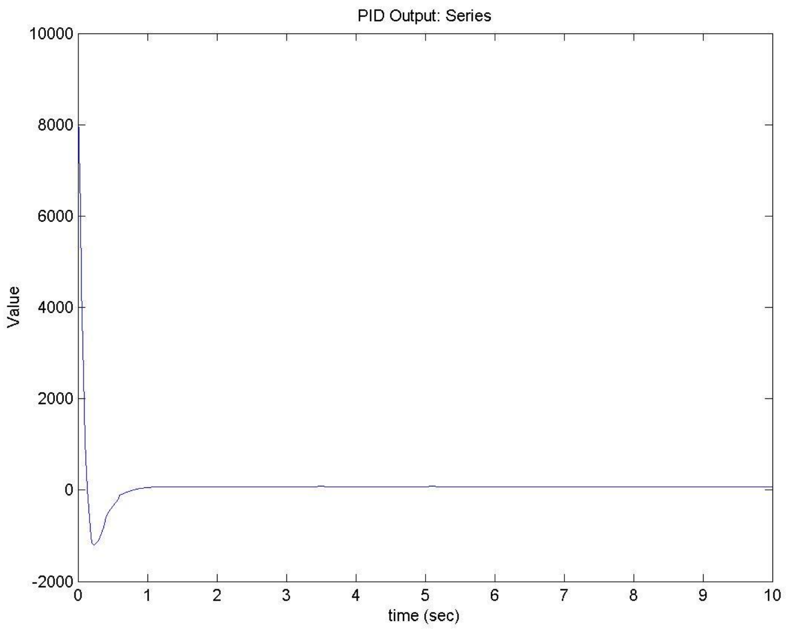


Figure 34: PID output, Series

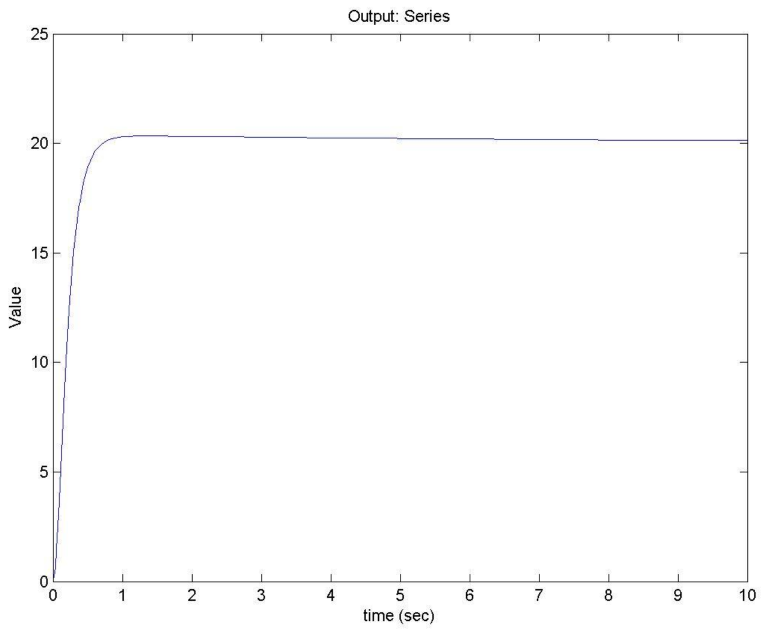


Figure 35: System output, Series

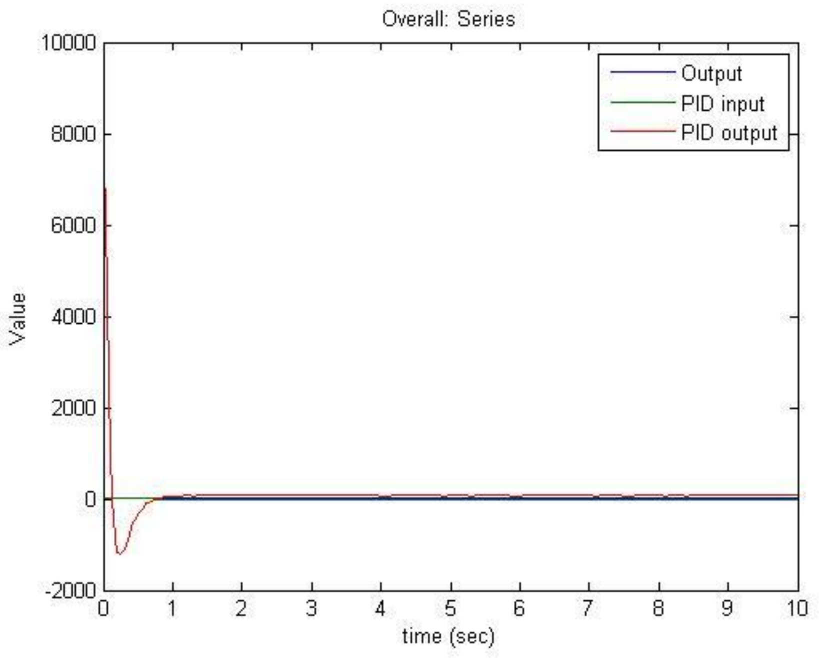


Figure 36: Overall, Series

As seen in the figures above the parallel and series PID implementations had the best results. Both the ideal and parallel PID implementations were under damped although it is much more obvious in the ideal implementation. The series implementation turned out to be either critically damped or over damped. From the results, we reasoned that it would be best to implement the series or parallel PID, preferably the series implementation since it had no overshoot.

6.2.7 Conversions

The overall control diagram is shown below.

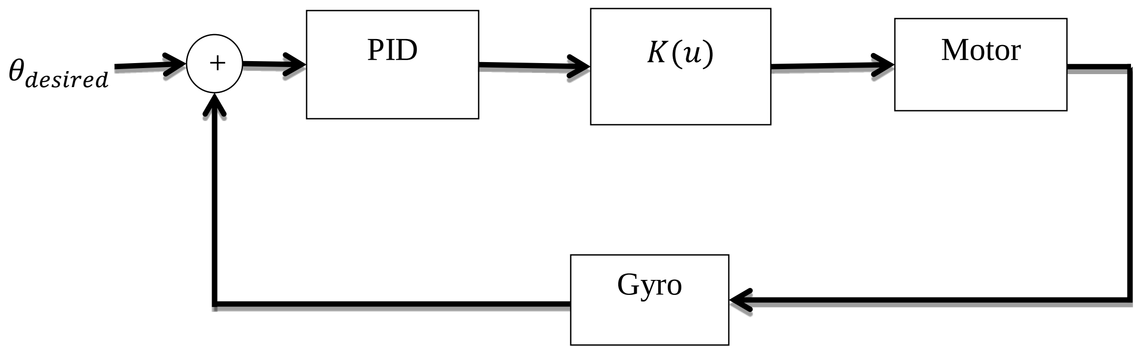


Figure 37: Control Diagram

Originally, we planned to have the motor speed as a direct input into the PID but we came across several problems. The only motor output we could measure easily was the angle of the quadcopter which is caused by the difference in thrusts on each motor, thus we could not use the motor speed as a feedback unless conversions ($K(u)$) were made. This would force us to convert the desired angle into motor speed. We then came across the problem of keeping the angle constant; when the actual angle reached the desired angle the control would still interpret it as an erroneous motor speed. To prevent this we moved the conversions after the PID. This way we would only have to convert once and the erroneous output would not occur. The conversions were derived by looking at the net forces and torques on the system as shown in the figure below.

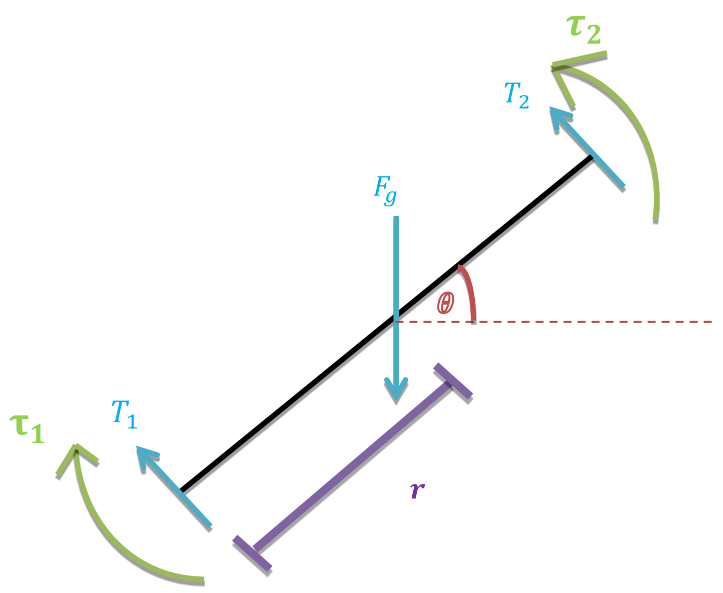


Figure 38: Quadcopter Free-Body Diagram

r: radius of arm, θ: angle of system relative to ground

τ_n: nth torque about center of system

T_n: nth motor trust

F_g: gravitational force

The net torque calculated,

$$\tau_{net}: \quad 0 = \tau_1 - \tau_2$$

$$= T_1 r - T_2 r$$

$$T_1 r = T_2 r$$

$$T_1 = T_2$$

Net forces in x and y directions,

$$F_x = T_{1x} + T_{2x} + T_{3x} + T_{4x} \qquad F_y = T_{1y} + T_{2y} + T_{3y} + T_{4y} - F_g$$

$$= \sum_{i=1}^{4} T_{ix} \qquad = \sum_{i=1}^{4} T_{iy} - F_g$$

To maintain a particular orientation all four motors have to produce the same thrust making the net torque about the center of the system zero. If the system is tilted along a particular axis then a net force will be produced that moves it along that same axis. If the system is horizontal relative to the ground then it will remain in place. To increase height the thrust on all four motors has to increase by the same amount. This is shown in the figure below[9].

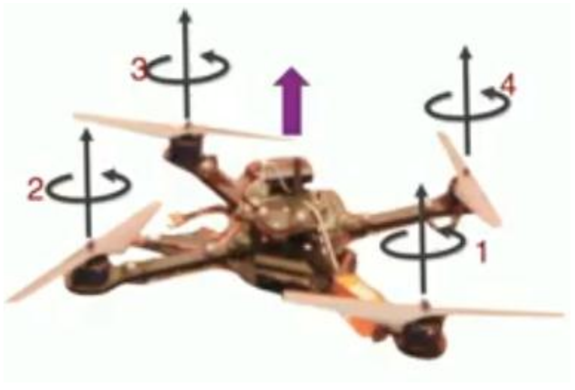


Figure 39: Quadcopter Z-axis movement

To change the orientation of the quadcopter a torque must be produced about the center of the system. This is achieved by changing the thrusts of the motors. Increasing the thrust of one motor while decreasing the thrust of the opposite motor produces a net torque. In the figure below, increasing the thrust of motor 4 and decreasing the thrust of motor 2 produces a counterclockwise torque that causes the quadcopter to change its roll angle.

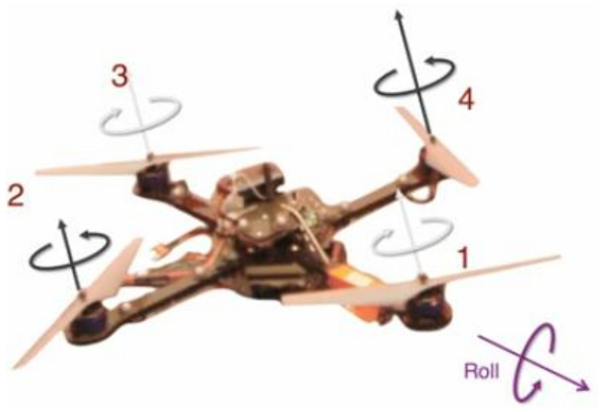


Figure 40: Quadcopter Axis Rotation

To rotate about the z-axis, or change the yaw angle, opposite pairs of motors must spin faster than the other pair as shown in the figure below.

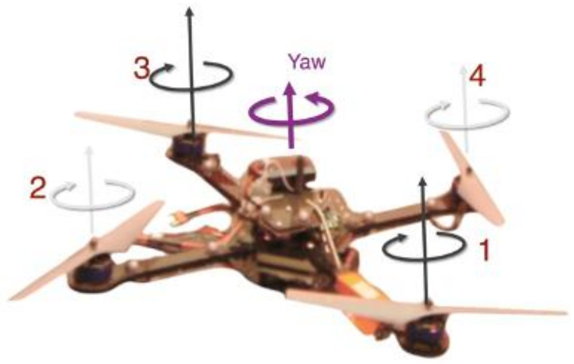


Figure 41: Quadcopter Yaw Rotation

To settle at a particular orientation the net torque has to be set to zero again. This means that all the motor speeds and thrusts have to be equal. The torques depend on the tangential force, which in turn depends on the tangential acceleration of the system about the center. The tangential acceleration can be represented as a function of angular acceleration as shown below.

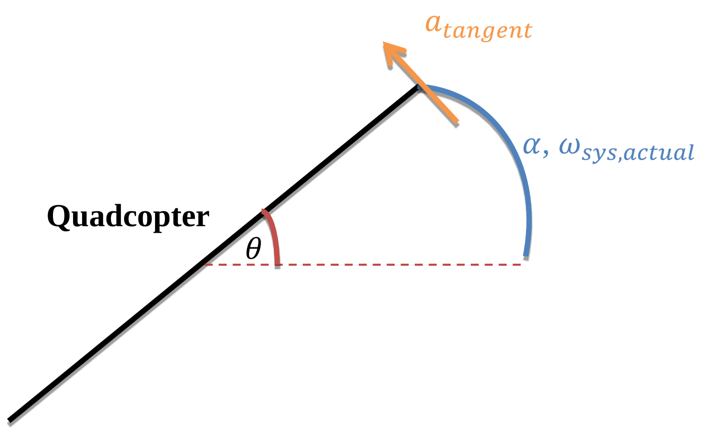


Figure 42: Tangential acceleration, velocity

$$a_{tangent} = r\alpha$$

We can then obtain the necessary angular acceleration and thrust with the following equations.

Measured angles: θ_n, θ_{n-1}

$$\omega_{sys,actual} = \frac{\theta_n - \theta_{n-1}}{t_n - t_{n-1}}, \qquad \omega_{sys,control} = \frac{u}{t_n - t_{n-1}}$$

$$\alpha = \frac{\omega_{sys,control} - \omega_{sys,actual}}{t_n - t_{n-1}}$$

$$T = ma_{tangent} = mr\alpha$$

Within the PID, the angle values are numerically integrated and differentiated as shown below:

$$\frac{d\theta}{dt} = \frac{\theta_n - \theta_{n-1}}{t_n - t_{n-1}}, \qquad \Theta_\mathrm{n} = \theta_n(t_n - t_{n-1}) + \Theta_{n-1}$$

The thrust of the motor is a function of the motor speed, which is typically given in the motor datasheet.

$$T = f(w_{motor})$$

$$\omega_{motor} = f^{-1}(T) = f^{-1}\left(ma_{tangent}\right) = f^{-1}(mr\alpha)$$

Thus, the motor speed ω_{motor} can be calculated with the angular acceleration of the system, which in turn is calculated by the current and previous angle readings. The speed of the motor would be increased/decreased according to the orientation desired. This would cause an acceleration to occur which in turn produces a torque about the center of the system. If the desired orientation is achieved then the change in angle would be zero causing the acceleration to be zero and preventing any additional torque. In this instance, the output of the control would just be the previous motor speed. Only the roll and pitch angles were considered for now since yaw control requires a more complicated calculation. Additionally, it is more critical to control roll and pitch than yaw for stability.

6.3 IR Detection System

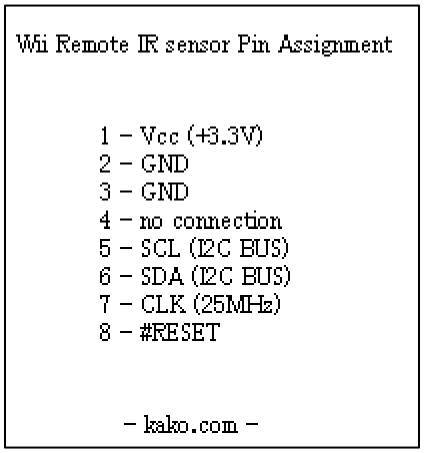
The IR Detection System can be broken down further into two subsystems, the IR Tracking Systems and the IR Communication System. Both systems are critical for the overall IR Detection System. An infrared camera is needed to determine the X and Y coordinates of a transmitter in relation to the quadcopter (when looking down at the ground). Depending on the camera and software used, it may not be possible to determine the information encoded in the transmitter's pulses. The camera might have a low sampling rate or require more complicated programming to extract the ID from the IR pulses.

While it was possible that we could have used a regular webcam and some image processing software, we found a much simpler solution. The Nintendo Wii Remote has an internal IR camera manufactured by Pixart [17]. Being a videogame controller, the Wii Remote is optimized for motion capture and point tracking. The IR camera is capable of tracking up to 4 infrared points. The camera then outputs the X and Y coordinates of the IR point detected, which was exactly what we needed for the quadcopter. We do not plan to test the quadcopter with too many transmitters, so four point tracking was sufficient. An additional benefit of the Pixart IR camera is the fact that it uses I2C communication. The I2C communication allows us to easily use a microcontroller read the data from the camera.

To decode the actual ID in the IR pulses, we decided that a dedicated IR receiver was the best solution. We already had a working IR pulse-width communicator from a previous project, so we planned to use it as part of our QUAD-S project. The IR communicator has a 4-bit input, which satisfies our 3-bit ID number with a package/transmitter flag.

6.3.1 IR Tracking System

Since the Pixart IR camera from the Wii Remote is not a readily available part, there were no data sheets to determine the pinout of the IR camera. We had to rely on online projects from people who had reverse-engineered the camera. Of these projects, we found three resources that were useful in determining how to use the Wii IR camera. The first resource is from a Japanese website written by Kako (translated into English using an online translator) [18]. Kako was able to reverse-engineer the Wii camera, including the pinout, registers where data is stored, and the algorithm to extract the desired data from these registers. Kako provides us with the pinout of the IR camera as shown in Figure 43.



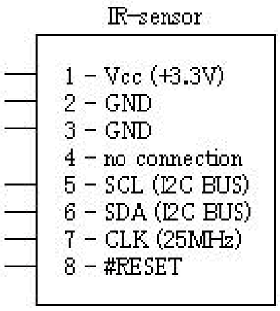
Wii Remote IR sensor Pin Assignment

1 – Vcc (+3.3V)
2 – GND
3 – GND
4 – no connection
5 – SCL (I2C BUS)
6 – SDA (I2C BUS)
7 – CLK (25MHz)
8 – #RESET

– kako.com –

Figure 43: IR Camera Pinout

From Figure 43, we see that the pinout from the IR Camera is fairly simple. Figure 44 below shows the actual pin numbers of the device, with the pinout next to it as reference.



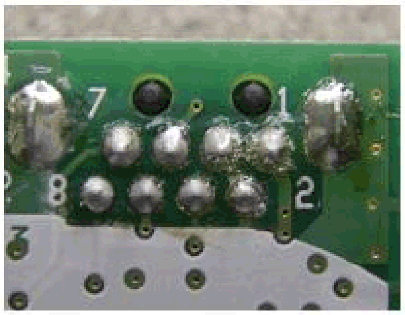


Figure 44: Pinout of Camera on actual circuit board

According to Kako's website, a 25 MHz clock is needed to drive the camera. However, according to Peter "RobotFreak" and Johnny Chung Lee, a clock signal of 20 MHz to 25 MHz is sufficient [19][20]. Since we had 20 MHz crystal oscillators readily available, we decided to use one to drive the camera. We used Kako's schematic in Figure 45 as a reference for designing our schematic.

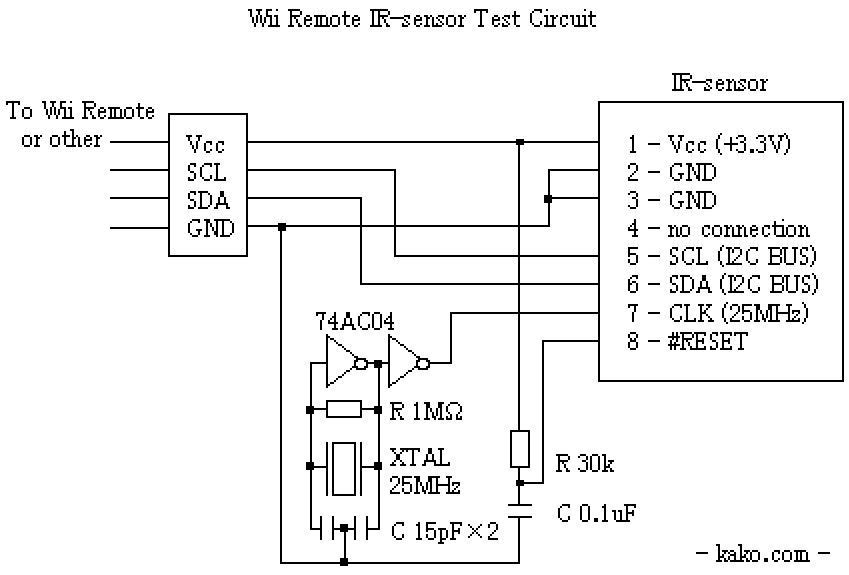


Figure 45: Recommended Schematic by Kako

We originally planned to use a PIC16F884 to communicate with the IR camera, but we found the code to be inefficient and we had trouble communicating with the camera during initial testing. Our initial schematic for the IR camera circuit is shown in Figure 46.

Our first attempt to extract the IR camera from a Wii remote circuit board resulted in failure. We managed to de-solder the camera, but the pins turned out to be too small to practically solder on, so we damaged the terminals in the process. After ordering a few more remotes, we decided to solder the wires to the camera while it was still on the circuit board, since there were bigger soldering pads to attach the wires to. The new schematic of the circuit using the Boarduino is shown in Figure 47.

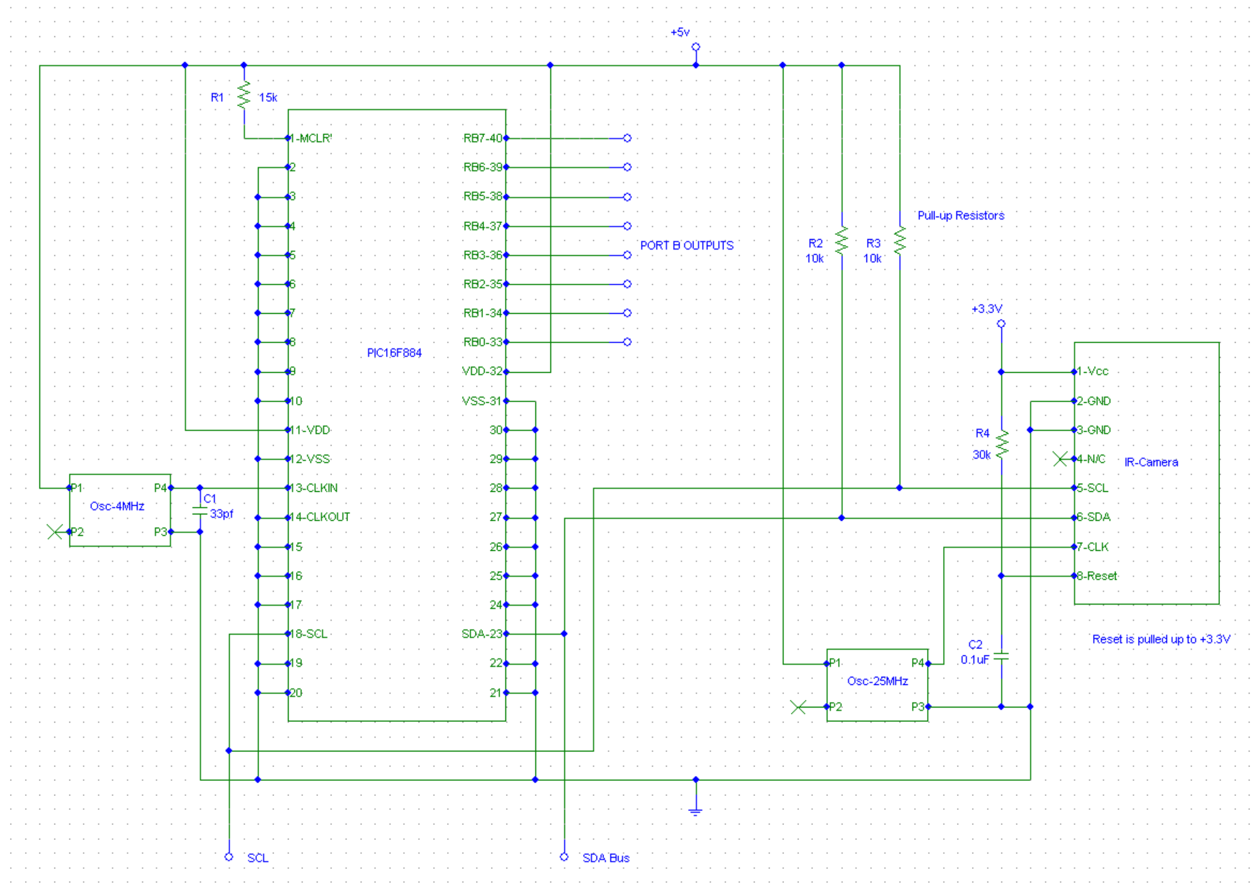


Figure 46: PIC IR Camera schematic

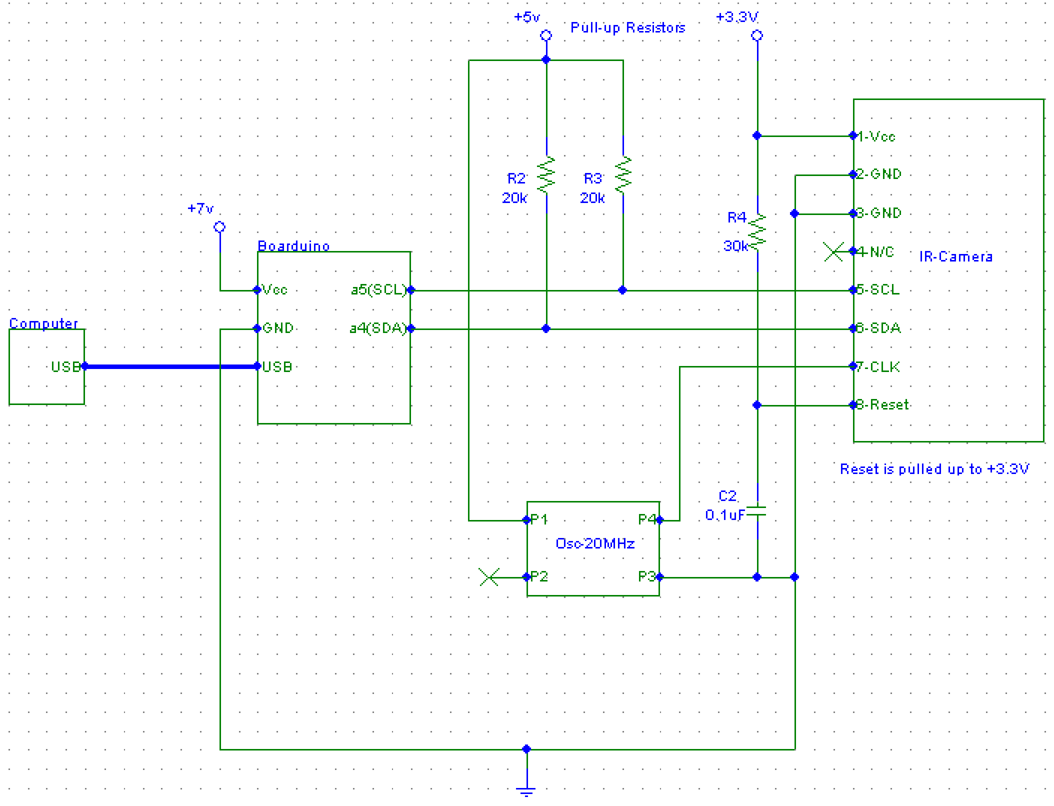


Figure 47: IR Camera schematic with Boarduino

We found Arduino code online that already implemented reading from the IR camera and displaying the X and Y coordinates to the Serial Monitor [21]. The Arduino code is shown in Appendix-1, Appendix-2, and Appendix-3.

We modified Arudino code from Steve Hobley that allowed us to read in the coordinates of the infrared sources measured by the IR camera [21]. We first modified his Wii library to calculate distance from the center of the camera's field of view. The Wii IR camera is capable of detecting point source coordinates with a 1024 x 768 resolution [22]. However, we found that the origin (0, 0) is actually measured from the bottom-left corner. Assuming that the camera is mounted at the center of the quadcopter, then all distances should be calculated from the point (512, 384). We took this point into consideration when using the distance formula to calculate the distance. In the main Arduino code, we made some modifications to the Serial output to make the data easy to read. In addition, we used a simple search to find the closest IR point on a 2D plane since the camera would be looking down at the ground when mounted on the quadcopter. The program would also print which action for the quadcopter to take in order to move toward the close IR point to investigate.

The main code reads the data from the camera and determines how many IR sources, referred to as "blobs", were detected, as well as the position of each source. The header files do the actual calculations of the X and Y coordinates from the data extracted from the camera registers. We modified these headers to also calculate the distance and save the value in the Blob structure. These distances could then be displayed along with the X and Y coordinates already stored in the structure.

6.3.2 IR Communication System

We decided to use an already functioning IR Communication system from a previous project. The IR transmitter has a 4-bit input. The microcontroller then encodes the four bits as a series of pulses and outputs the pulse-width signal to an IR LED. The pulse width timings are shown below in Table 2.

Table 2: Pulse Timing Diagram

Reset pulse width:	29.60 ms
High '1' pulse width:	14.80 ms
Low '0' pulse width:	7.400 ms
Delay	8.900 ms

The IR receiver then looks for a reset pulse before measuring the width of each data pulse and determining if the bit is high or low. The microcontroller on the receiver then outputs the 4-bit code to four LEDs. This four-bit output will eventually be used as inputs to the control system so that the ID and package/target identifier can be recorded

6.4 Selection of Components

Below is a list of components we used this semester and a brief summary of why we selected them:

- MPU-6050 accelerometer/gyroscope

  From our research, we determined that having both a gyroscope and accelerometer would result in a more stable system overall.

- Wii Remote

  The Wii Remote is the only known way to obtain the Pixart IR Camera, which is not sold publically. The camera has four IR point tracking and uses the simple I2C communication protocol.

- Boarduino (w/ ATMEGA 168-20PU)

  The Boarduino is an open-source Arduino-compatible development board that can be placed directly in a breadboard. It uses the ATMEGA 168-20PU chip. Much of the code we modified was made for Arduino, so we used this board to easily prototype our I2C devices on the breadboard. Another factor for its use was that it was already in the possession of one of our group members and would therefore be of no cost to us.

- Arduino Uno

  Like the Boarduino, we needed another Arduino microcontroller to test the code we found from the MPU-6050 sensor manufacturer. The Boarduino above did not have sufficient memory to hold all the libraries required to test our sensor, so we bought the Arduino Uno, which was locally available and had twice the memory of the ATMEGA 168-20PU.

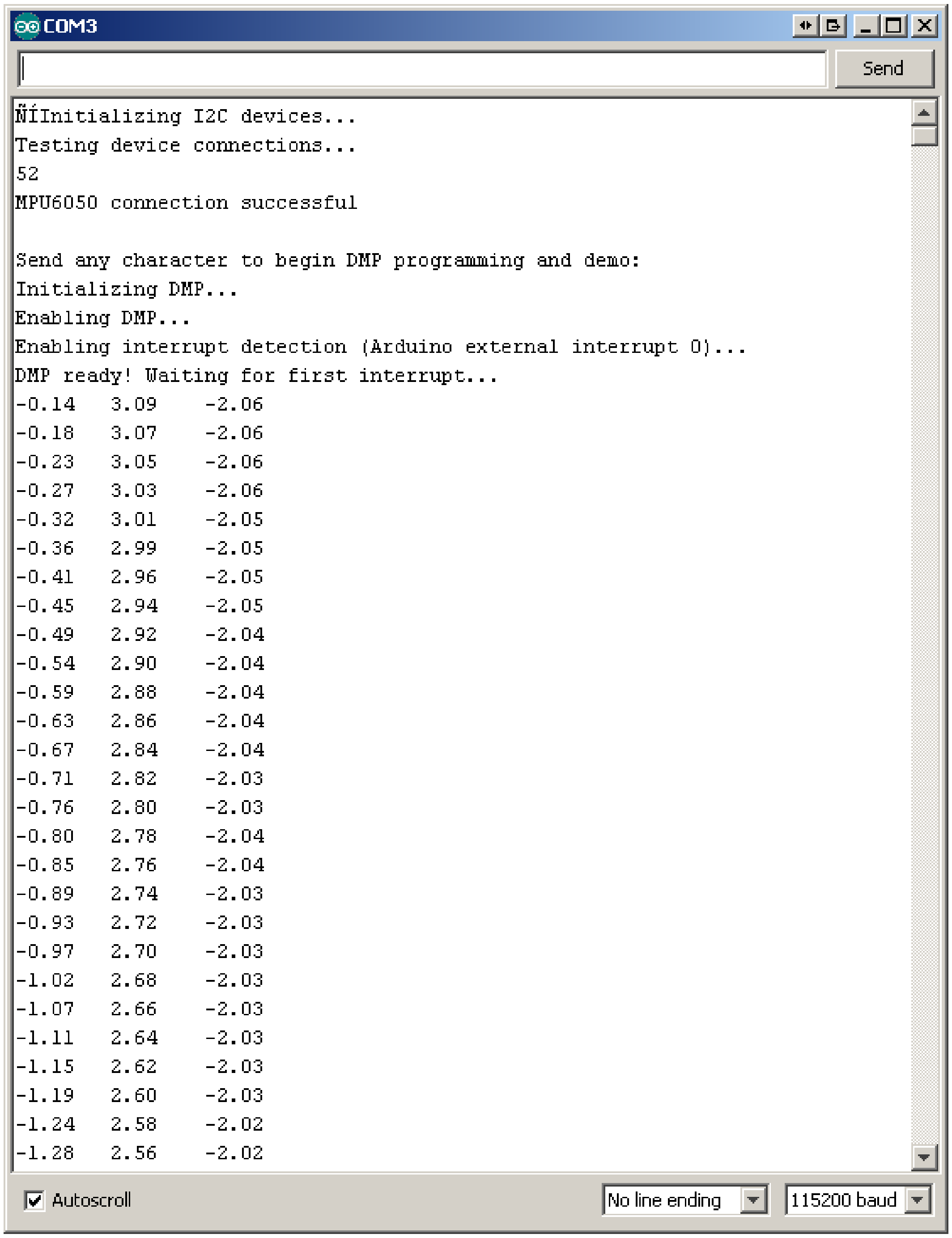
- PING))) Ultrasonic Distance Sensor

  We mainly chose the PING Ultrasonic Distance Sensor because it was parted from an older project, so it did not cost us anything. In addition, we found that the sensor had a simple input and output pin that outputs a pulse width related to the distance measured. This pulse width would be fairly simple to measure.

7 Design Measurements and Evaluation

7.1 Stability System

For the stability system, we performed some tests and measurements on the Accel/Gyro sensor. At the beginning of the Arduino command prompt, the program had to be started or initialized by sending any character then the data was displayed showing Euler angles (ypr) or the acceleration (xyz) as called from the Arduino code. Some print screens of this measurement are shown in Figure 48.

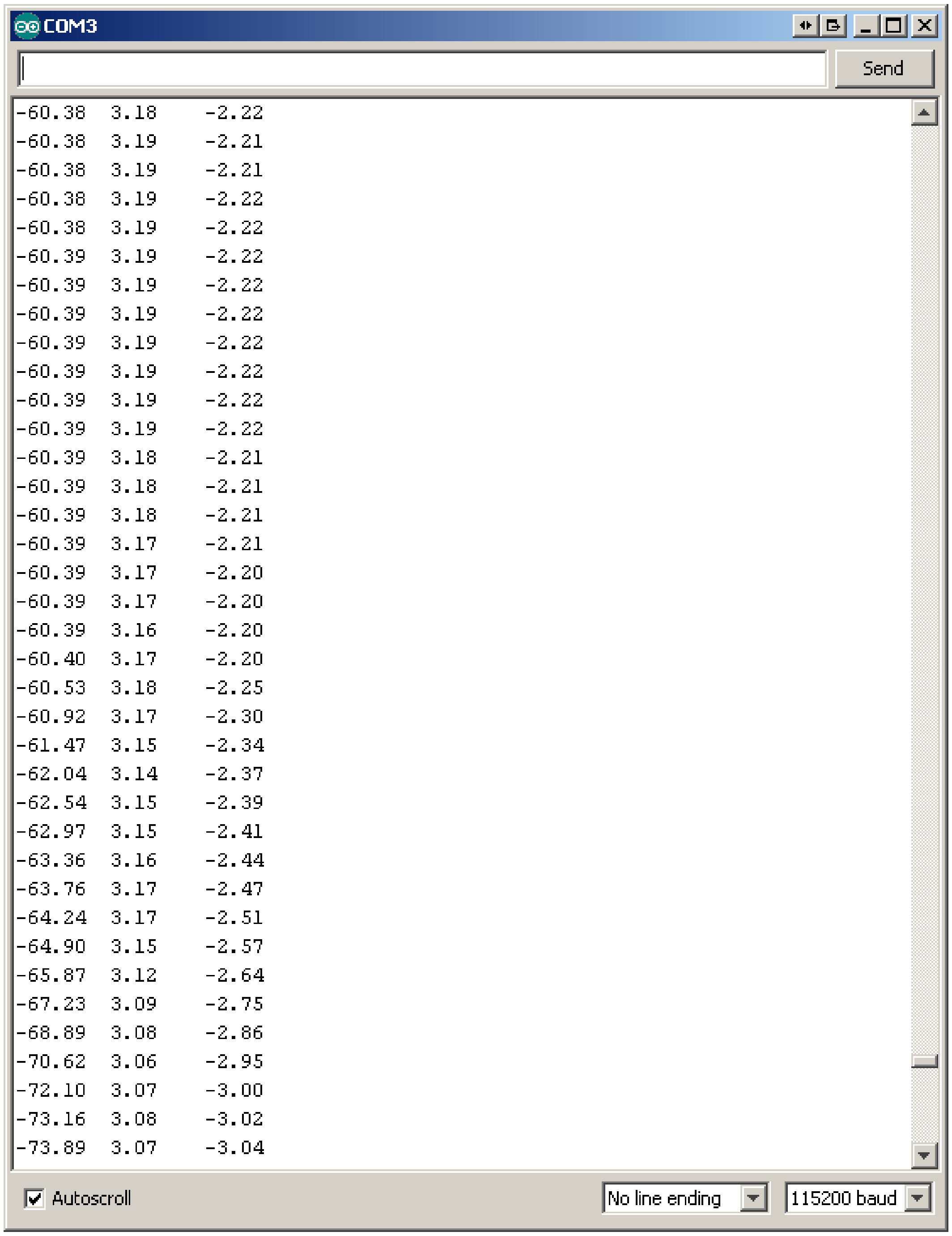


```
ÑÍInitializing I2C devices...
Testing device connections...
52
MPU6050 connection successful

Send any character to begin DMP programming and demo:
Initializing DMP...
Enabling DMP...
Enabling interrupt detection (Arduino external interrupt 0)...
DMP ready! Waiting for first interrupt...
-0.14   3.09    -2.06
-0.18   3.07    -2.06
-0.23   3.05    -2.06
-0.27   3.03    -2.06
-0.32   3.01    -2.05
-0.36   2.99    -2.05
-0.41   2.96    -2.05
-0.45   2.94    -2.05
-0.49   2.92    -2.04
-0.54   2.90    -2.04
-0.59   2.88    -2.04
-0.63   2.86    -2.04
-0.67   2.84    -2.04
-0.71   2.82    -2.03
-0.76   2.80    -2.03
-0.80   2.78    -2.04
-0.85   2.76    -2.04
-0.89   2.74    -2.03
-0.93   2.72    -2.03
-0.97   2.70    -2.03
-1.02   2.68    -2.03
-1.07   2.66    -2.03
-1.11   2.64    -2.03
-1.15   2.62    -2.03
-1.19   2.60    -2.03
-1.24   2.58    -2.02
-1.28   2.56    -2.02
```

[✓] Autoscroll   No line ending   115200 baud

Figure 48: Serial Monitor output for Yaw-Pitch-Roll angles



```
-60.38  3.18    -2.22
-60.38  3.19    -2.21
-60.38  3.19    -2.21
-60.38  3.19    -2.22
-60.38  3.19    -2.22
-60.39  3.19    -2.22
-60.39  3.19    -2.22
-60.39  3.19    -2.22
-60.39  3.19    -2.22
-60.39  3.19    -2.22
-60.39  3.19    -2.22
-60.39  3.19    -2.22
-60.39  3.18    -2.21
-60.39  3.18    -2.21
-60.39  3.18    -2.21
-60.39  3.17    -2.21
-60.39  3.17    -2.20
-60.39  3.17    -2.20
-60.39  3.16    -2.20
-60.40  3.17    -2.20
-60.53  3.18    -2.25
-60.92  3.17    -2.30
-61.47  3.15    -2.34
-62.04  3.14    -2.37
-62.54  3.15    -2.39
-62.97  3.15    -2.41
-63.36  3.16    -2.44
-63.76  3.17    -2.47
-64.24  3.17    -2.51
-64.90  3.15    -2.57
-65.87  3.12    -2.64
-67.23  3.09    -2.75
-68.89  3.08    -2.86
-70.62  3.06    -2.95
-72.10  3.07    -3.00
-73.16  3.08    -3.02
-73.89  3.07    -3.04
```

Figure 49: Yaw-Pitch-Roll angles (continued)

From the first test, we noted that the sensor took about 13 seconds to stabilize this was important to note for our data taking tests. Once the sensor stabilized, we began moving the sensor in the different orientations i.e. yaw, pitch and roll directions and observed the columns change; the first column in the print screens represented the yaw, then the second pitch and the third roll.

With the data from now quantized, we researched ways to interface that serial data coming from the accel/gyro sensor through the Arduino Uno Board into MATLAB so we could plot those points and have a better visual of the results of our tests.

We noticed that the data coming from the Arduino Board was coming in serial so we looked for “help serial” in MATLAB and found some helpful commands that would perform the task we wanted, the commands to call on the serial data were *fopen()* and *fscanf()*.

Once we knew how to interface MATLAB and the Arduino, we develop a MATLAB script that would call the serial data from the Arduino, our code can be seen in its entirety in Appendix-4. We then ran our script to be able to see the data in the command window of MATLAB and started moving the sensor in the yaw, pitch, and roll directions. Our resulting data for the Euler angles is shown in Figure 50 through Figure 52.

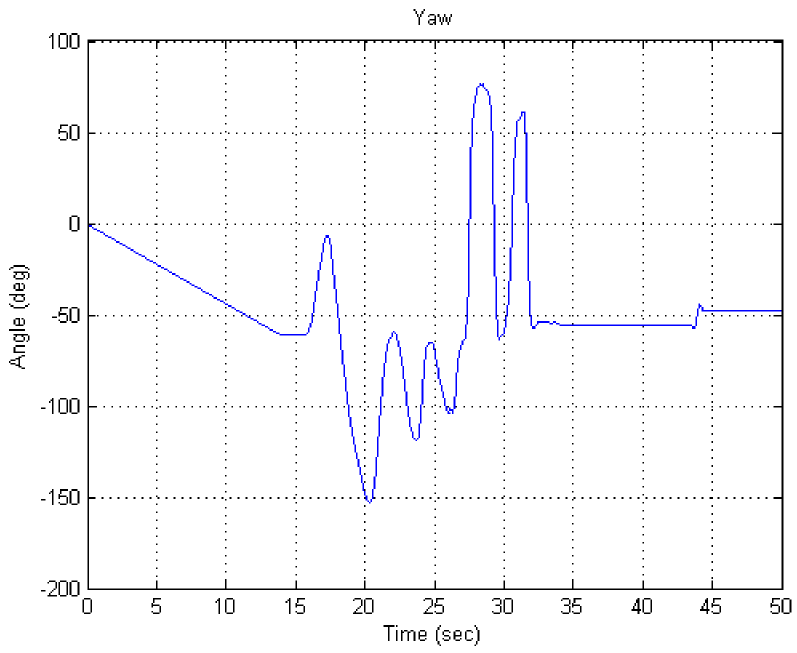


Figure 50: Yaw Angle Plot

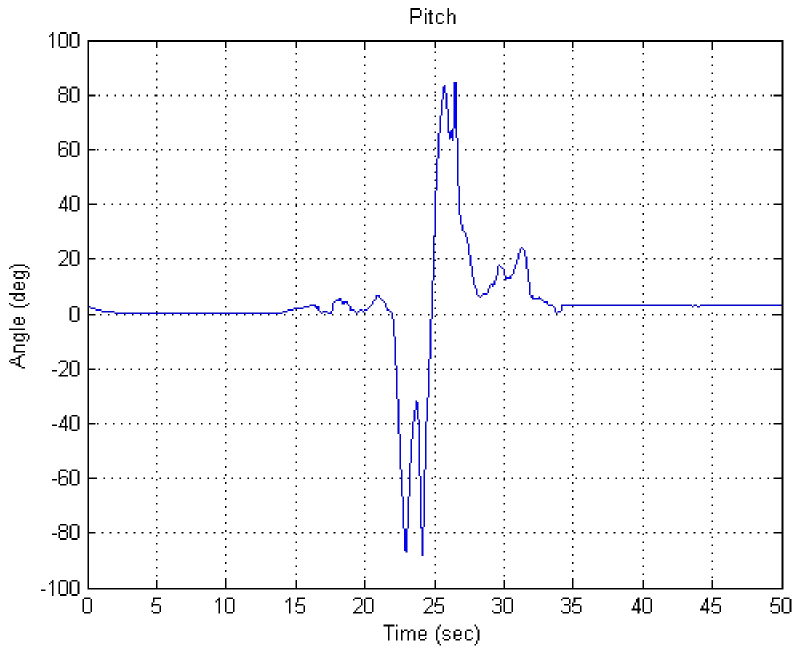


Figure 51: Pitch Angle Plot

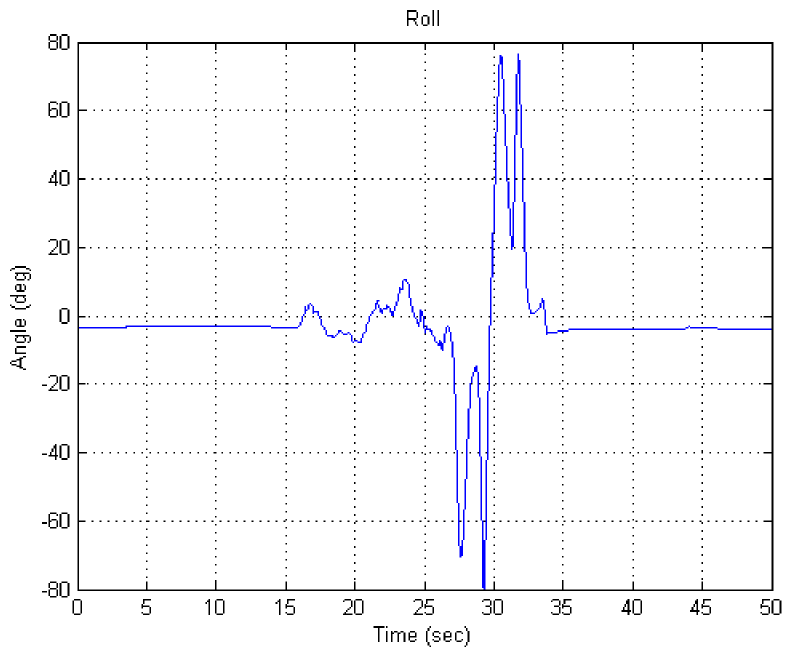


Figure 52: Roll Angle Plot

From our results we were able to confirm what we had mentioned in our early tests with just the Arduino serial window, the sensor needed some time (around 13 seconds) to stabilize but only for the yaw angle. Once the angles stabilized, we began by moving the sensor in the yaw direction and that is how we observed the change in degrees for the yaw angle (Spikes on Figure 50). The next step was to move the sensor in the Pitch direction followed by the Roll direction, as with the yaw direction, we observed the same changes (spikes) in angular position. Finally, we just left the sensor immobile and that is why towards the end the response is constant.

We then tested the acceleration in the same fashion, resulting in the plots in Figure 53 through Figure 55.

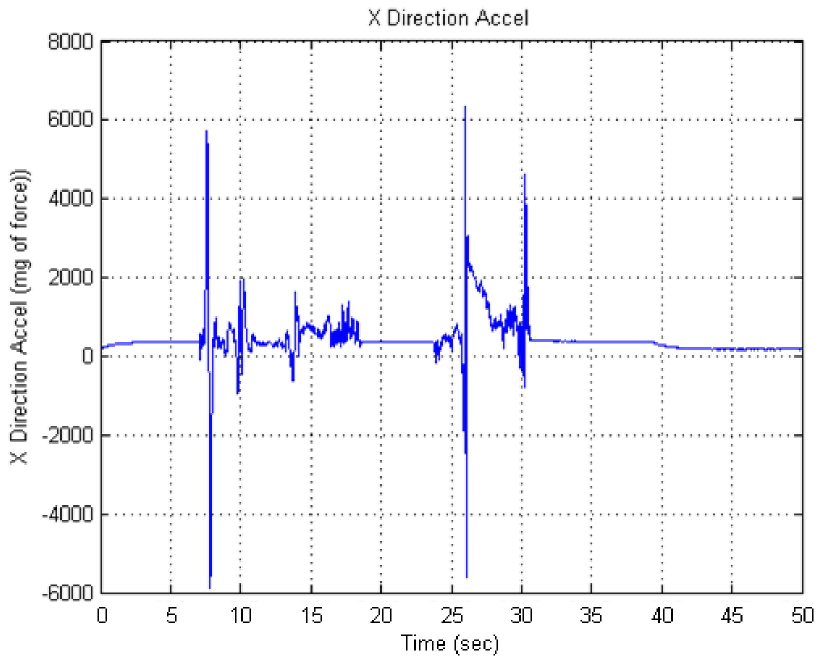


Figure 53: X Acceleration

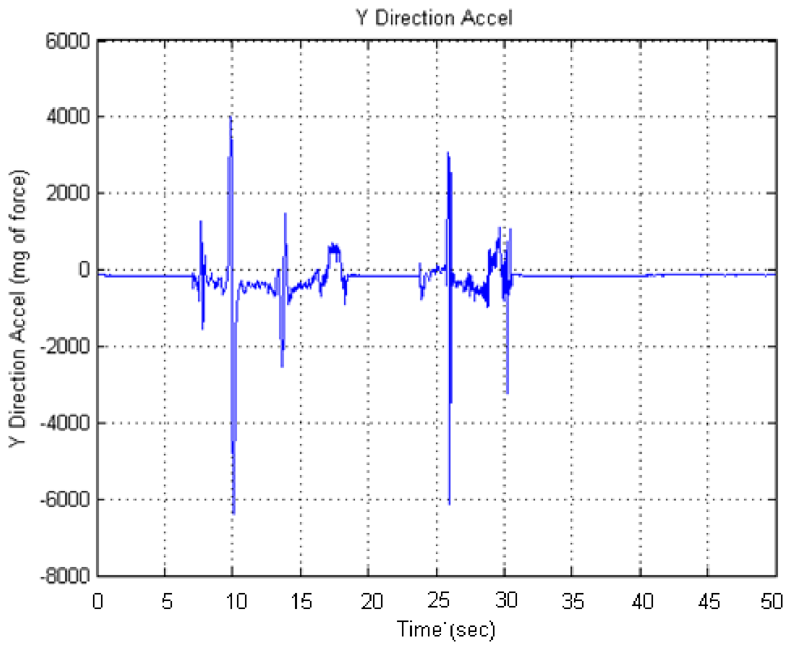


Figure 54: Y Acceleration

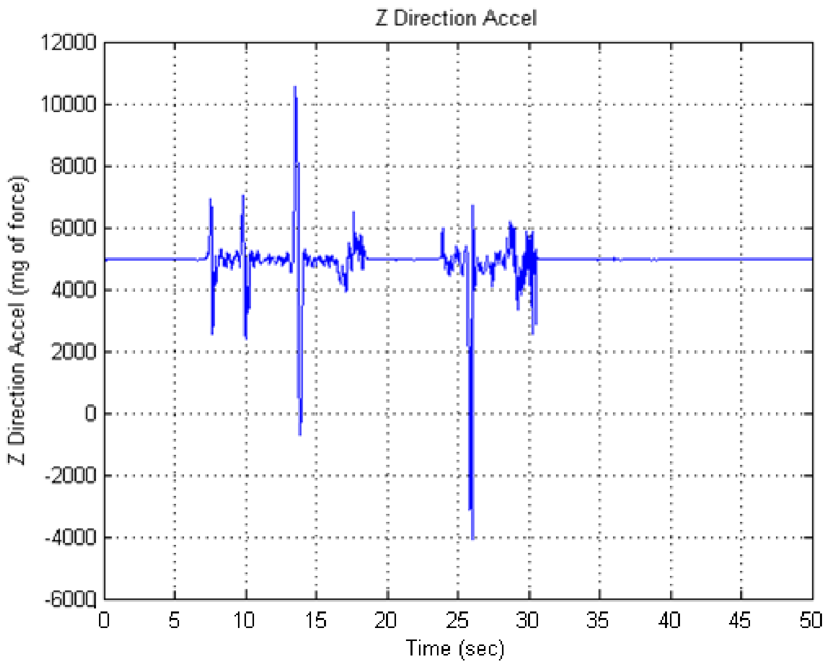


Figure 55: Z Acceleration

The results from Figure 53 through Figure 55 show the linear acceleration (given in milli-g's of force according to the sensor's datasheet [1]). This test was performed by first moving the sensor in the + x direction with a sudden jerk, this is why we observed the first spike having some positive acceleration and then as it slows down it attains negative acceleration. We then proceeded by moving it in the +y direction with the same spiking but this time attaining acceleration in the y direction then we moved it in the + z direction reflected on the first spike in Figure 6. Finally, a test of free fall was made by just letting the sensor fall down from about a foot and then catching it while in midair, this response was observed in the second negative spike of Figure 55.

7.2 Control Systems

Both the parallel and the series PID implementations were tested with the gyroscope sensor. The PID and overall control was programmed in MATLAB. The Arduino received rotational data from the sensor, which was then sent to MATLAB via serial interface. The angles and control signal were updated in real time by the PID control code. The first file, SensorInput.m (Appendix-5), is a modification of the sensor serial interfacing program developed earlier. The motor control was implemented within the sensor-reading loop by calling the second file, AngleControl.m (Appendix-6). This file calls the PID program and performs conversions from the angle measurements to desired motor speed. The third file, PID.m (Appendix-7), runs the actual PID calculations. The results are shown in the figures below. Motors 1 and 3 responded to a change in roll while motors 2 and 4 responded to a change in pitch.

7.2.1 Parallel

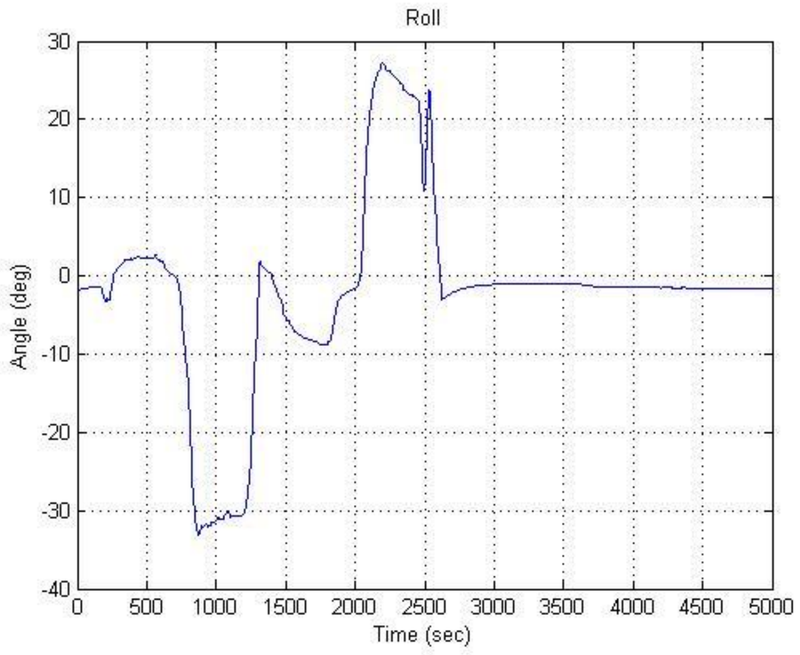


Figure 56: Roll Angle

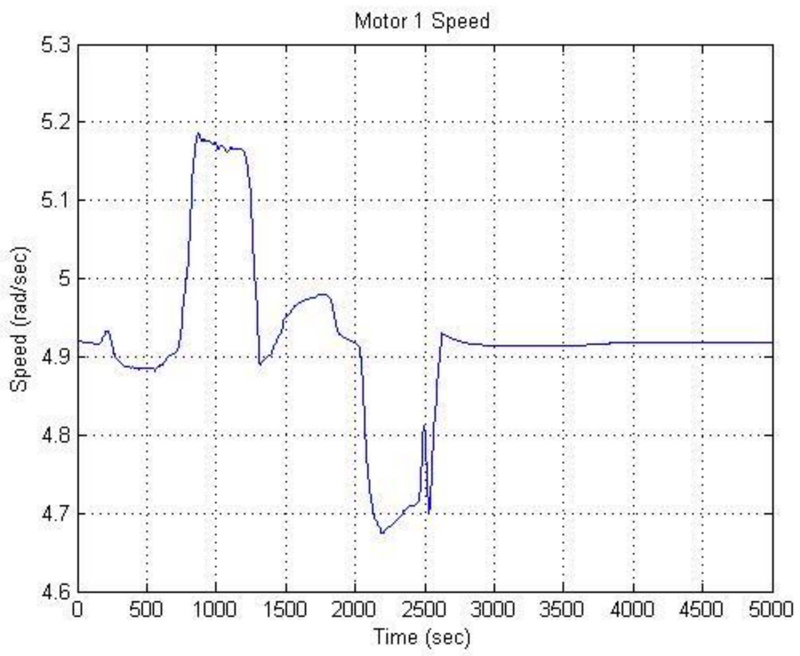


Figure 57: Motor 1 Speed

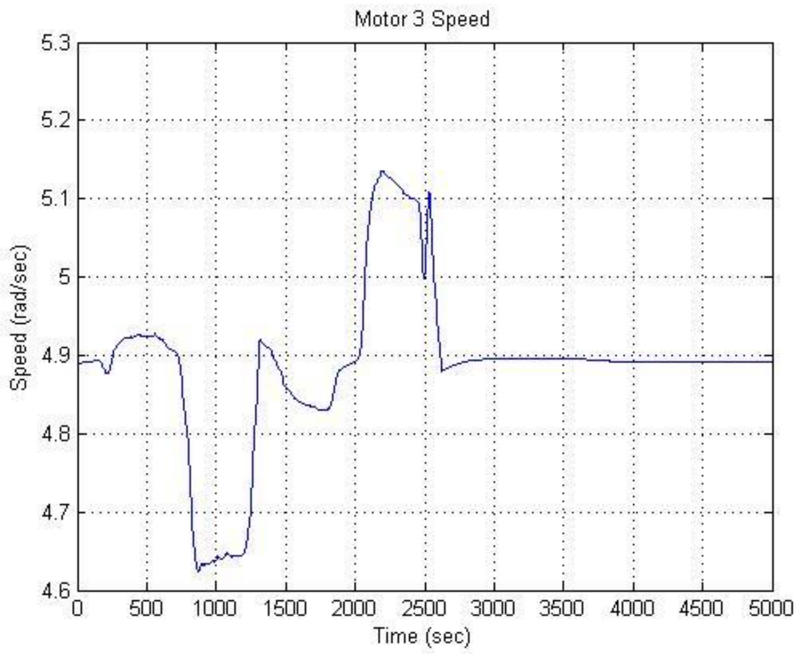


Figure 58: Motor 3 Speed

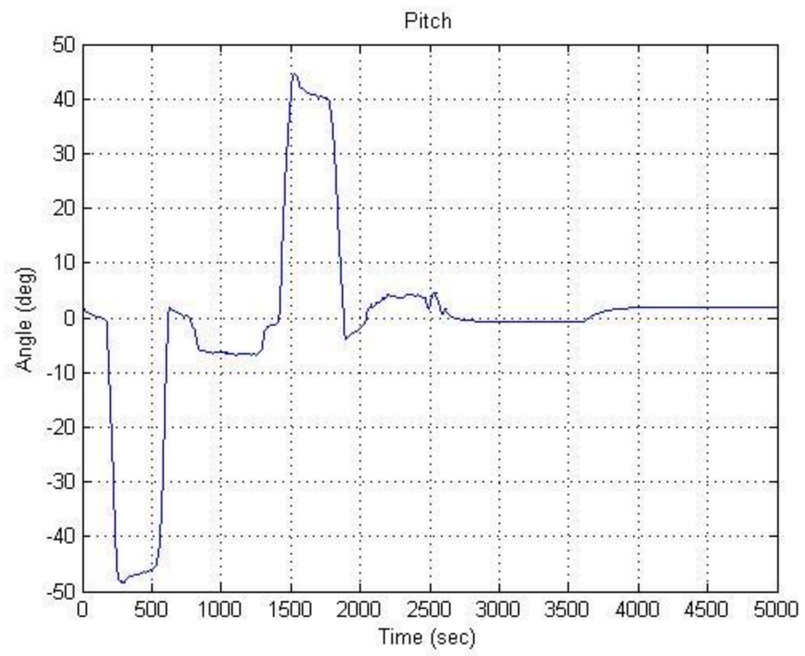


Figure 59: Pitch Angle

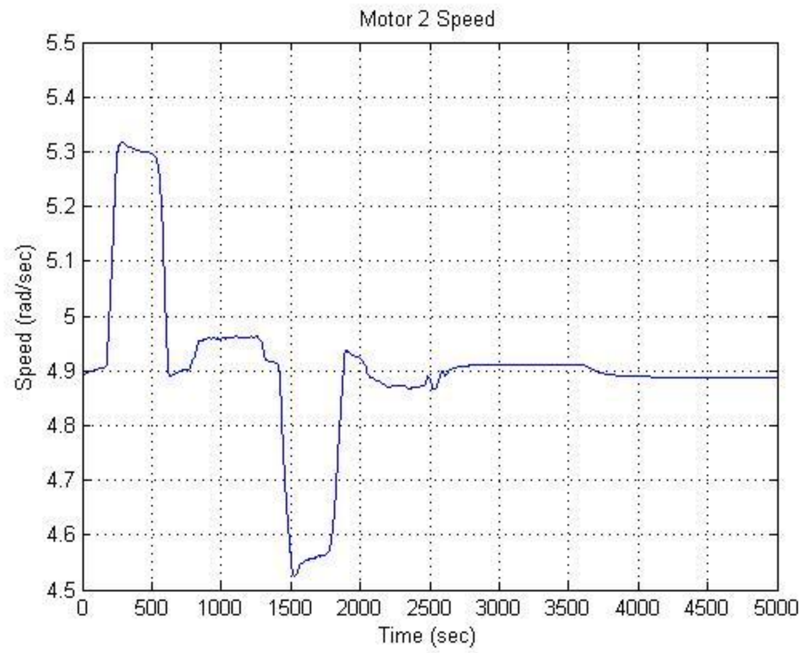


Figure 60: Motor 2 Speed

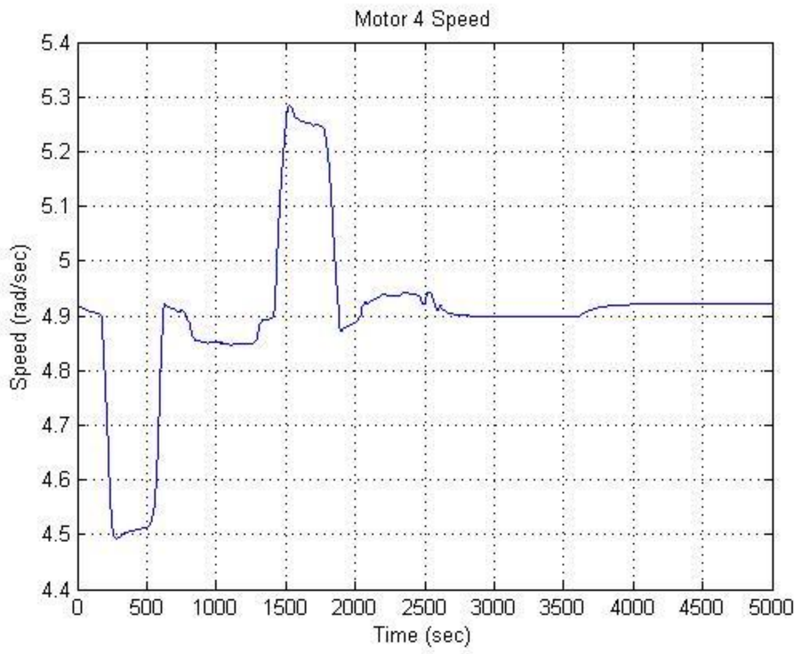


Figure 61: Motor 4 Speed

7.2.2 Series

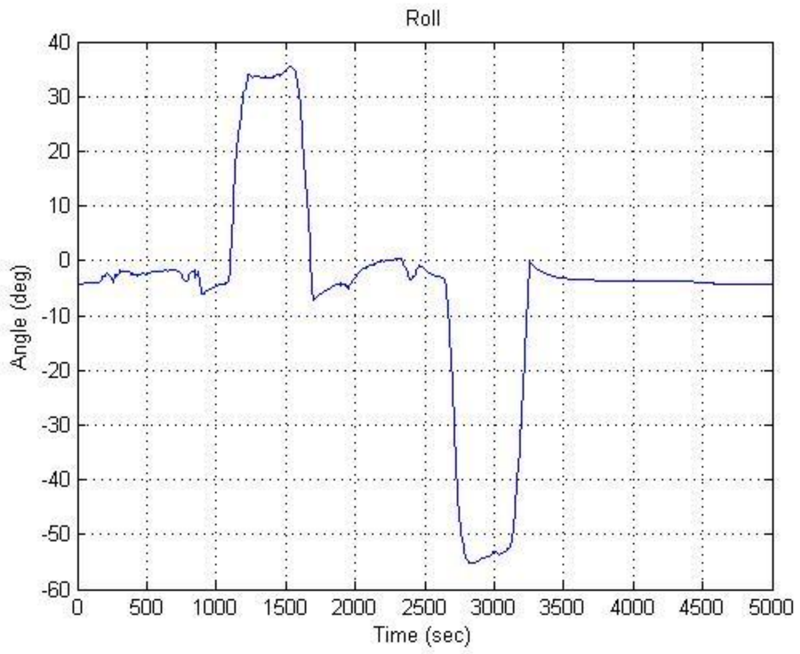


Figure 62: Roll Angle

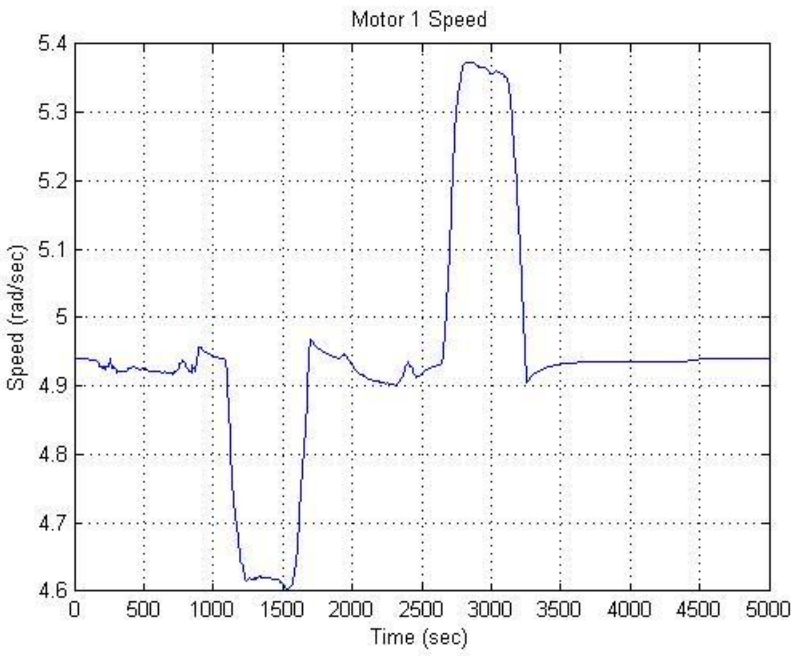


Figure 63: Motor 1 Speed

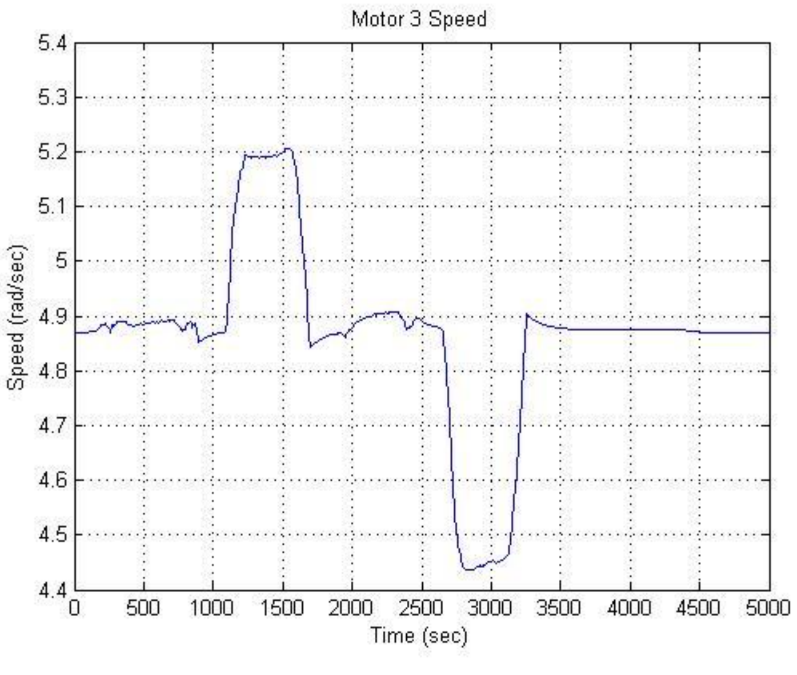


Figure 64: Motor 3 Speed

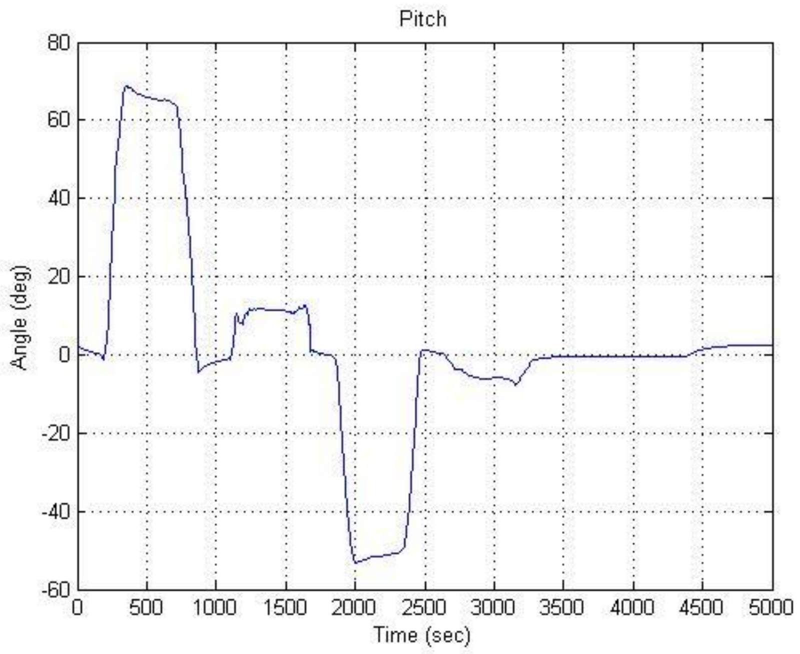


Figure 65: Pitch Angle

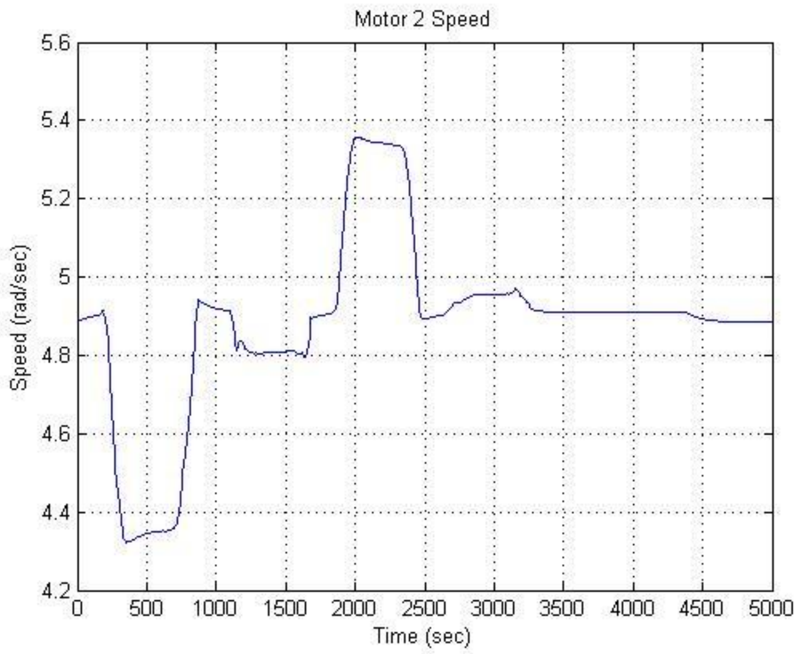


Figure 66: Motor 2 Speed

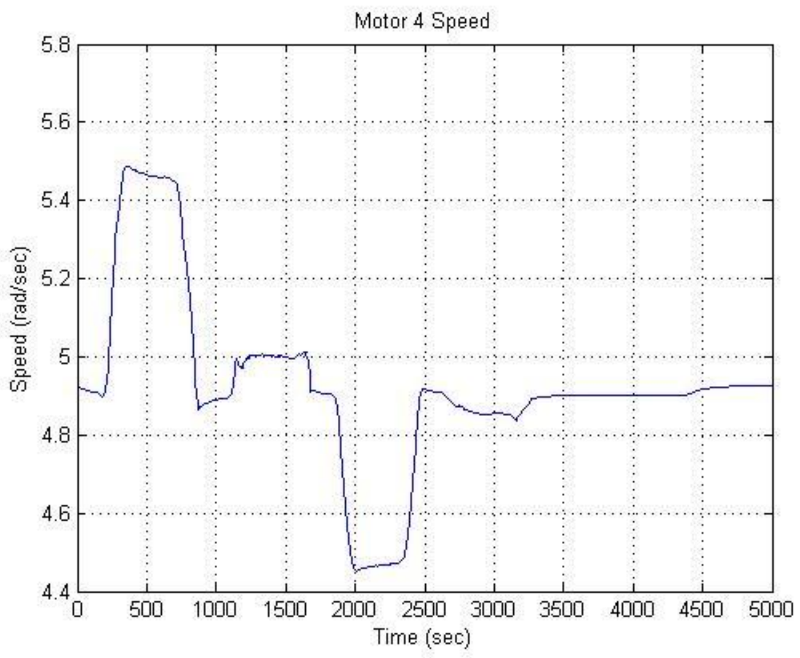
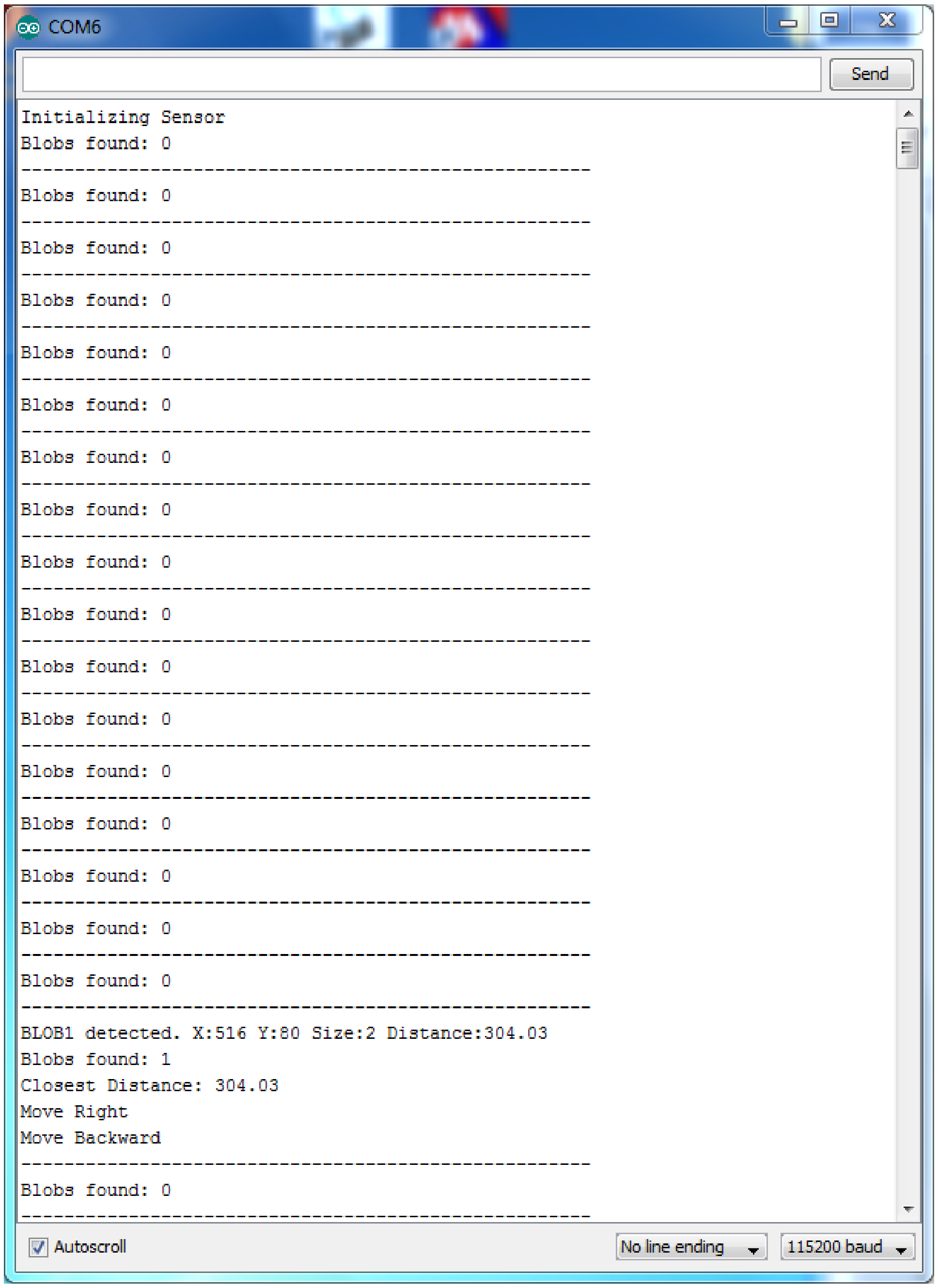


Figure 67: Motor 4 Speed

As seen above, the motors responded appropriately when changes in angle were applied. Motors 1 and 3 had opposite responses and they responded correctly to change in roll. The same can be said for motors 2 and 4 corresponding to pitch. There were no significant differences between both PID implementations since no actual motors were used. The PID only displayed what the motors should do instead of what was actually going on. Only the control signals for each implementation were tested.

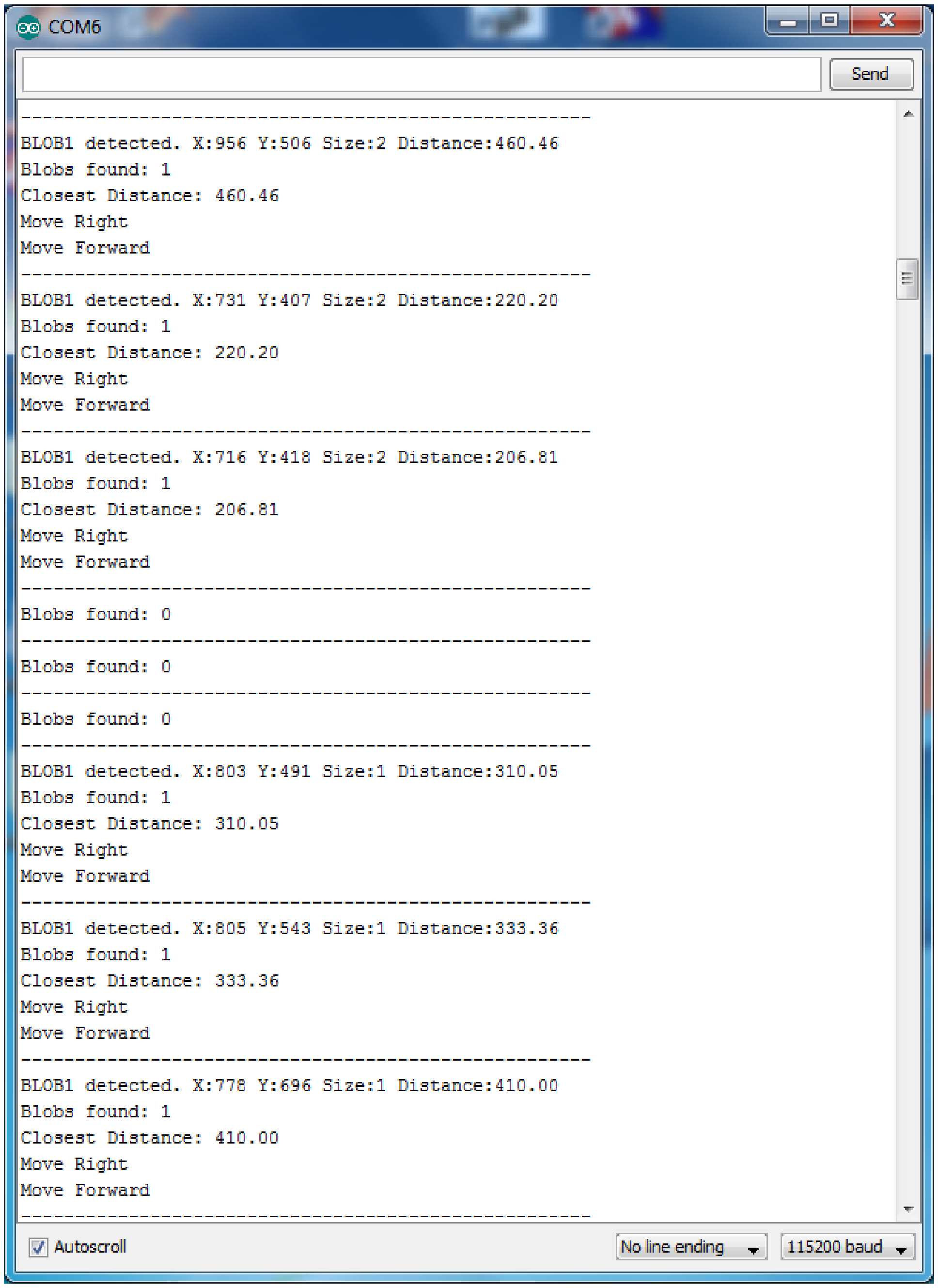
7.3 IR Detection System

We ran a few tests to track IR LEDs using the Arduino program. In the first test, we used a television remote with a sufficiently powerful IR LED to see if we could track its position. Television remotes transmit data using a series of pulses. Since we plan to use IR transmitters that also use pulses to transmit data, we figured the TV remote would be a good starting test to see if the pulses interfere with the camera's ability to detect its position. Some of our test results are shown in Figure 68 and Figure 69.



```
Initializing Sensor
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
BLOB1 detected. X:516 Y:80 Size:2 Distance:304.03
Blobs found: 1
Closest Distance: 304.03
Move Right
Move Backward
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
```

Figure 68: IR Camera Program Start and Detecting one point



```
-----------------------------------------------------
BLOB1 detected. X:956 Y:506 Size:2 Distance:460.46
Blobs found: 1
Closest Distance: 460.46
Move Right
Move Forward
-----------------------------------------------------
BLOB1 detected. X:731 Y:407 Size:2 Distance:220.20
Blobs found: 1
Closest Distance: 220.20
Move Right
Move Forward
-----------------------------------------------------
BLOB1 detected. X:716 Y:418 Size:2 Distance:206.81
Blobs found: 1
Closest Distance: 206.81
Move Right
Move Forward
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
BLOB1 detected. X:803 Y:491 Size:1 Distance:310.05
Blobs found: 1
Closest Distance: 310.05
Move Right
Move Forward
-----------------------------------------------------
BLOB1 detected. X:805 Y:543 Size:1 Distance:333.36
Blobs found: 1
Closest Distance: 333.36
Move Right
Move Forward
-----------------------------------------------------
BLOB1 detected. X:778 Y:696 Size:1 Distance:410.00
Blobs found: 1
Closest Distance: 410.00
Move Right
Move Forward
-----------------------------------------------------
```

[✓] Autoscroll   No line ending   115200 baud

Figure 69: IR Camera program detecting the TV remote at various position

We see that the IR camera was successfully able to detect the TV remote, even if it uses pulses. The pulsing of the IR LED did not seem to have any effects on the camera, such as repeated pauses then detections. We also verified the distance was correct based on the position of the point detected and its distance from the center point at (512, 384). We were able to detected the TV remote at a distance of about 7 feet and we expect it to be able to detect points up to 10 feet, which would be a sufficient flying height for the quadcopter.

If the detected point had an X component greater than 512, then the point is to the right of the quadcopter, so we expect the quadcopter to have to move right. Likewise, if the X component is less than 512, then the point is to the left of the quadcopter. If the Y component of the detected point is greater than 384, it is in front of the quadcopter, so the quadcopter must move forward. If the Y component is less than 384, then the point is behind the quadcopter, so it must move backward. The diagram below shows how we determine which direction the quadcopter has to move.

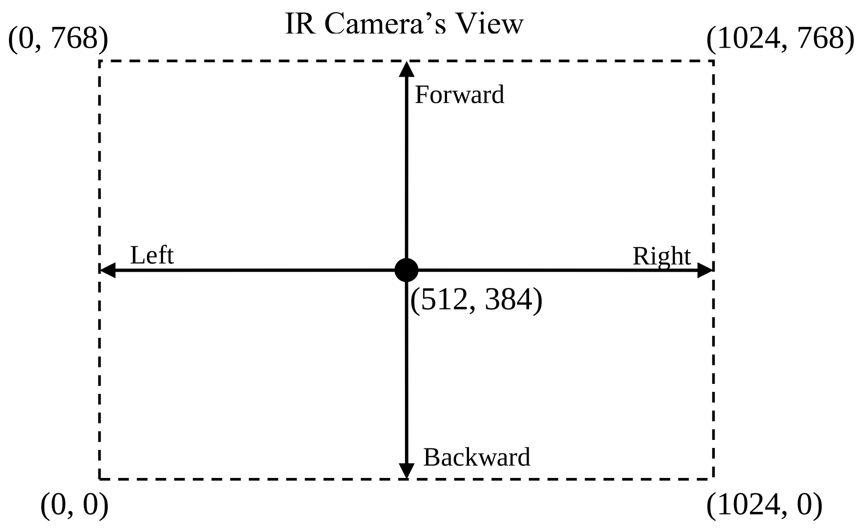
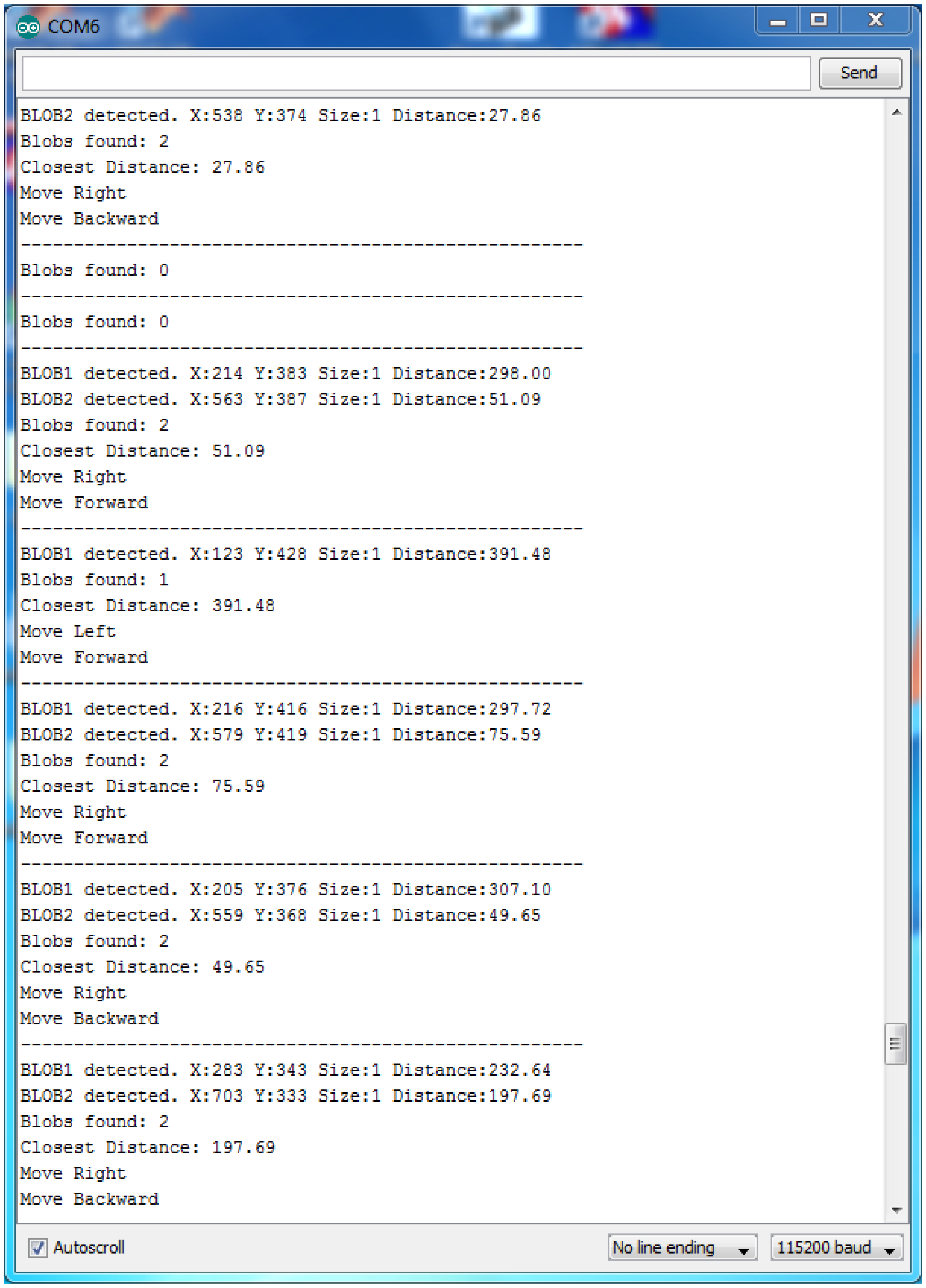


Figure 70: IR Camera View

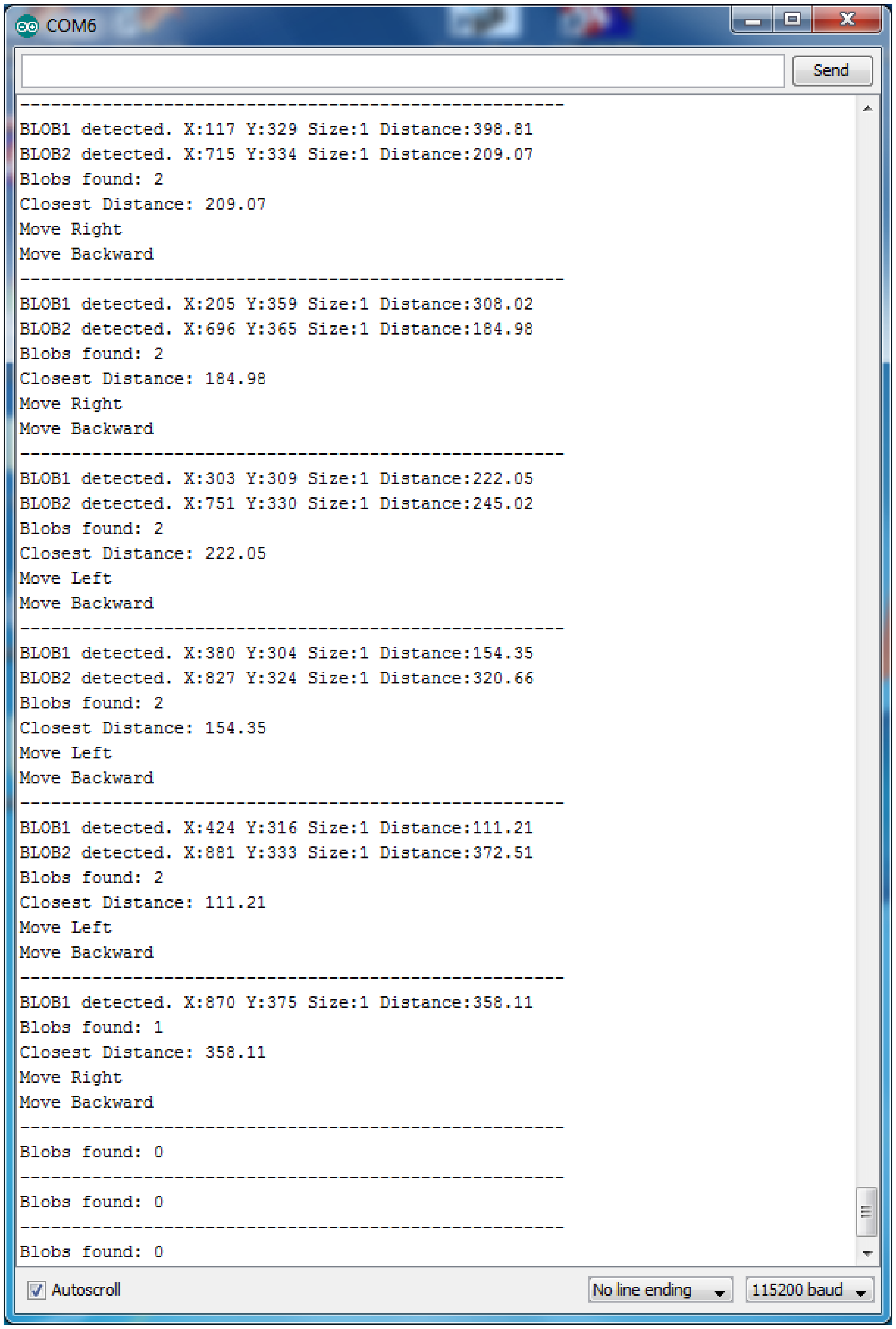
In the next test, we set up two IR LEDs spread apart on a breadboard to test if the camera could detect more than one point and determine which was the closest. The results of this test are shown in Figure 71, Figure 72, and Figure 73.



```
BLOB2 detected. X:538 Y:374 Size:1 Distance:27.86
Blobs found: 2
Closest Distance: 27.86
Move Right
Move Backward
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
BLOB1 detected. X:214 Y:383 Size:1 Distance:298.00
BLOB2 detected. X:563 Y:387 Size:1 Distance:51.09
Blobs found: 2
Closest Distance: 51.09
Move Right
Move Forward
-----------------------------------------------------
BLOB1 detected. X:123 Y:428 Size:1 Distance:391.48
Blobs found: 1
Closest Distance: 391.48
Move Left
Move Forward
-----------------------------------------------------
BLOB1 detected. X:216 Y:416 Size:1 Distance:297.72
BLOB2 detected. X:579 Y:419 Size:1 Distance:75.59
Blobs found: 2
Closest Distance: 75.59
Move Right
Move Forward
-----------------------------------------------------
BLOB1 detected. X:205 Y:376 Size:1 Distance:307.10
BLOB2 detected. X:559 Y:368 Size:1 Distance:49.65
Blobs found: 2
Closest Distance: 49.65
Move Right
Move Backward
-----------------------------------------------------
BLOB1 detected. X:283 Y:343 Size:1 Distance:232.64
BLOB2 detected. X:703 Y:333 Size:1 Distance:197.69
Blobs found: 2
Closest Distance: 197.69
Move Right
Move Backward
```

☑ Autoscroll   No line ending   115200 baud

Figure 71: Program Test with 2 IR LEDs

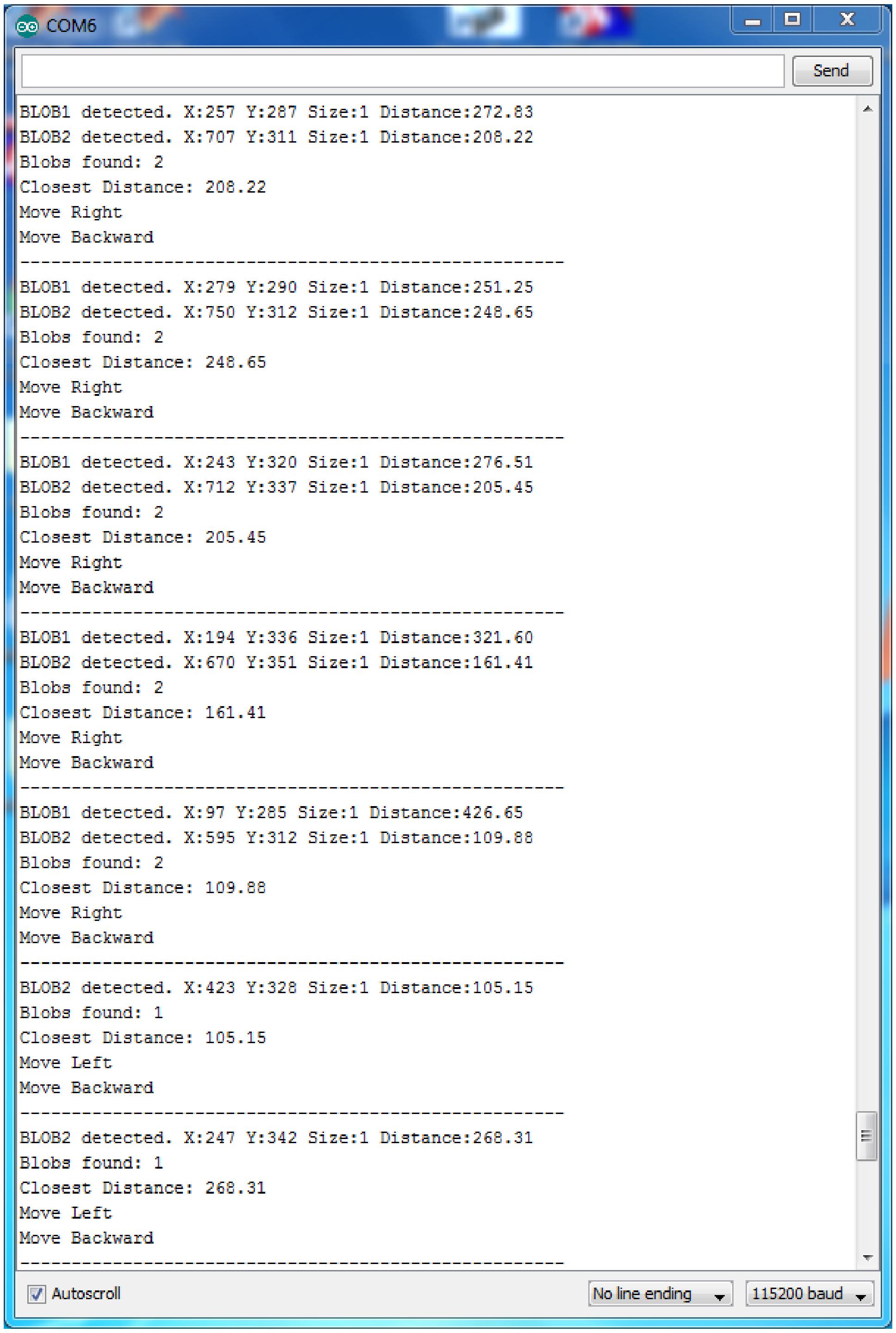

COM6

Send

```
-----------------------------------------------------
BLOB1 detected. X:117 Y:329 Size:1 Distance:398.81
BLOB2 detected. X:715 Y:334 Size:1 Distance:209.07
Blobs found: 2
Closest Distance: 209.07
Move Right
Move Backward
-----------------------------------------------------
BLOB1 detected. X:205 Y:359 Size:1 Distance:308.02
BLOB2 detected. X:696 Y:365 Size:1 Distance:184.98
Blobs found: 2
Closest Distance: 184.98
Move Right
Move Backward
-----------------------------------------------------
BLOB1 detected. X:303 Y:309 Size:1 Distance:222.05
BLOB2 detected. X:751 Y:330 Size:1 Distance:245.02
Blobs found: 2
Closest Distance: 222.05
Move Left
Move Backward
-----------------------------------------------------
BLOB1 detected. X:380 Y:304 Size:1 Distance:154.35
BLOB2 detected. X:827 Y:324 Size:1 Distance:320.66
Blobs found: 2
Closest Distance: 154.35
Move Left
Move Backward
-----------------------------------------------------
BLOB1 detected. X:424 Y:316 Size:1 Distance:111.21
BLOB2 detected. X:881 Y:333 Size:1 Distance:372.51
Blobs found: 2
Closest Distance: 111.21
Move Left
Move Backward
-----------------------------------------------------
BLOB1 detected. X:870 Y:375 Size:1 Distance:358.11
Blobs found: 1
Closest Distance: 358.11
Move Right
Move Backward
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
-----------------------------------------------------
Blobs found: 0
```

☑ Autoscroll   No line ending   115200 baud

Figure 72: Test with 2 IR LEDs (continued)



```
BLOB1 detected. X:257 Y:287 Size:1 Distance:272.83
BLOB2 detected. X:707 Y:311 Size:1 Distance:208.22
Blobs found: 2
Closest Distance: 208.22
Move Right
Move Backward
-----------------------------------------------------
BLOB1 detected. X:279 Y:290 Size:1 Distance:251.25
BLOB2 detected. X:750 Y:312 Size:1 Distance:248.65
Blobs found: 2
Closest Distance: 248.65
Move Right
Move Backward
-----------------------------------------------------
BLOB1 detected. X:243 Y:320 Size:1 Distance:276.51
BLOB2 detected. X:712 Y:337 Size:1 Distance:205.45
Blobs found: 2
Closest Distance: 205.45
Move Right
Move Backward
-----------------------------------------------------
BLOB1 detected. X:194 Y:336 Size:1 Distance:321.60
BLOB2 detected. X:670 Y:351 Size:1 Distance:161.41
Blobs found: 2
Closest Distance: 161.41
Move Right
Move Backward
-----------------------------------------------------
BLOB1 detected. X:97 Y:285 Size:1 Distance:426.65
BLOB2 detected. X:595 Y:312 Size:1 Distance:109.88
Blobs found: 2
Closest Distance: 109.88
Move Right
Move Backward
-----------------------------------------------------
BLOB2 detected. X:423 Y:328 Size:1 Distance:105.15
Blobs found: 1
Closest Distance: 105.15
Move Left
Move Backward
-----------------------------------------------------
BLOB2 detected. X:247 Y:342 Size:1 Distance:268.31
Blobs found: 1
Closest Distance: 268.31
Move Left
Move Backward
-----------------------------------------------------
```

Autoscroll [✓] No line ending 115200 baud

Figure 73: Test with 2 IR LEDs (cont.)

Like the first test, we see that the program was able to detect the position of both IR LEDs and calculate the distance to each point. The program then determines which point is closer and what action the quadcopter should take to move toward that closest point. The main difficulty we had with this test was with the IR LEDs we were using. They have a very narrow viewing angle, making it difficult to direct them at the camera at greater distances. We plan to use high-intensity IR LEDs with the widest viewing angle possible in the final implementation. Another plan we have is to use multiple IR LEDs bunched together to provide better multi-directional transmission to increase the chances that the camera can see the transmitter. Overall we are satisfied with these intial tests of the IR camera.

8 Cost estimate

Table 3 shows the initial cost estimate for the QUAD-S

Table 3: Overall Project Cost Estimate

Part	**Price**
Frame Parts	$20
Battery	$20
Microcontroller (PIC)	$10
Accelerometer / Gyroscope	$40
DC Brushed Motors (or Brushless)	$40
Propellers	$10
IR Detection Parts	$20
Magnets	$5
IR Proximity Sensor	$60
Sonar Sensor	$22
Servo	$10
Miscellaneous (Replacements, etc.)	$40
TOTAL:	**$297**

In Table 4, we see the actual cost-to-date list for this semester. The entries in red denote parts we decided were not essential for completion of the critical systems for this semester. We plan to actually build the frame and mount the motors and propellers the second semester.

Table 4: Cost-to-Date List

Part	Price	Paid Price	
Frame Parts	$20		
Battery	$20		
Microcontroller	$10	$37.88	
Accelerometer / Gyroscope	$40	$45.09	
DC Brushed Motors (or Brushless)	$40		
Propellers	$10		
IR Detection Parts (Wii Remote)	$20	$25.48	
Magnets	$5		
IR Proximity Sensor	$60		
Sonar Sensor	$22	$0.00	
Servo	$10		
Miscellaneous (Replacements, etc.)	$40		Remaining
TOTAL:	**$297**	**$108.45**	**$188.55**

From Table 4, we see that we only spent about $100 dollars of the $300 budget. We spent more on the microcontrollers because we had to buy an Arduino Uno to replace the original PIC microcontrollers we first attempted to use. The extra expenses on the microcontroller were balanced by the acquisition of a free Sonar Sensor and some of the money from the miscellaneous section.

Table 5 below shows a more detailed breakdown of the part costs and the stores from which we obtained them. We had to purchase more Wii Remotes because our first attempt to extract the IR camera from the circuit board resulted in damage to the camera. We bought four more Wii Remotes as a lot sold for parts. The Ultrasonic Sensor was parted from previous projects, but is available from the manufacturer Parallax for $29.99.

Table 5: Detailed Cost Analysis

Date	Part	Part Number	Store	Price per part	Number of components	Shipping/Tax	Part Total	In Production	Sustainable	ROHS Compliant
9/18	PING))) Ultrasonic Distance Sensor	#28015	N/A	FREE	1	$0.00	$0.00	Yes	Yes	Yes
9/25	Accelerometer / Gyro Breakout MPU-6050	SEN-11028	www.sparkfun.com	$39.95	1	$3.64	$45.09	Yes	Yes	Yes
9/25	Breakaway Headers	PRT-00116	www.sparkfun.com	$1.50	1			Yes	Yes	Yes
9/30	Wii Remote (used)	RVL-003	Amazon.com	$9.89	1	$3.99	$13.88	Yes	Yes[1]	No
11/2	Boarduino (ATMEGA 168-20PU)	ID: 91	www.adafruit.com	FREE	1	$0.00	$0.00	Yes	Yes	No[2]
10/7	4 Wii Remotes (used, as-is)	RVL-003	eBay.com	$1.40	4	$6.00	$11.60	Yes	Yes[1]	No
11/2	Arduino Uno	276-128	Radio Shack	$34.99	1	$2.89	$37.88	Yes	Yes	Yes
						TOTAL	$108.45			

1. The IR Camera is sustainable as long as the Wii Remote remains in production
2. There was no indication on the manufacturer's website that the overall development board was ROHS compliant. Similar products have an ROHS seal on the description; the Boarduino did not have one. Since the user is required to assemble the product, the manufacturer might take into consideration lead-based soldering being used, such as we did. The ATMEGA 168-20PU itself is ROHS compliant.

9 Ethical and Political Impact

9.1 Health & Safety

With the nature of the quadcopter, we must take into consideration some safety issues. With four propellers spinning at a fast rate, there is potential for some cuts to humans and animals, especially when handling the quadcopter. Being a flying device, there is also possibilities of the quadcopter falling out of the air and landing on people, especially if the quadcopter loses control. We plan to use wide, clear areas for testing, as well as possibly tether the quadcopter to the ground. If necessary, we will wear hard hats when testing the quadcopter.

According to the Federal Aviation Administration, allows government agencies such as police to "operate unmanned aircraft weighing 4.4 pounds or less" with a high limit of 400 feet above the ground and within line of sight [23]. Our test would be more of a commercial/hobbyist flight, so it does not seem like we are allowed the same permission to fly as the government agencies. While we do not expect for our quadcopter to fly very high, we will fly it indoors if necessary as to not conflict with FAA law.

9.2 Environmental

Most of the components we used this semester were ROHS compliant except for the Boarduino and the Wii Remotes. Concerning the Boarduino, there was no indication on the manufacturer's website that the overall development board was ROHS compliant. Similar products have an ROHS seal on the description; the Boarduino did not have one. Since the user is required to assemble the product, the manufacturer might take into consideration lead-based soldering being used, such as we did. The ATMEGA168-20PU chip used by the Boarduino is ROHS compliant. Considering that eventually we wish to embed all of our microcontrollers, then we will no longer need the Boarduino development board, only the ATMEGA168-20PU chip. This means that the systems using this chip should remain ROHS compliant assuming that no other components have already made it non-compliant.

Since the Wii Remote is a consumer product, it has been difficult to determine if the device itself is ROHS compliant. In addition, the Pixart company that makes the IR camera itself does not have a publically available datasheet for the camera since it is only manufactured for Nintendo. Nintendo states that they comply with ROHS, as well as other waste and hazardous substance management, in their 2012 Corporate Social Responsibility (CSR) Report [24]. However, the RVL-003 model of the Wii Remote we used was manufactured in 2006 [25]. In the past, the company itself has received low environmental impact scores, but the validity of such reports can also be argued [26]. Due to the lack of ROHS information on the Wii Remote, we have deemed it to be non-ROHS.

9.3 Political Impact

Our device could possibly be used as surveillance by government facilities, which may cause privacy issues. However, if used for its actual purpose, it can actually be used by the government to help with disasters.

9.4 Manufacturability, Reliability, and Sustainability

As seen from our cost analysis, all of the components we used are still in production. The only difficulty as far as Manufacturability and Sustainability is the fact that the IR camera can only be extracted from a Wii Remote. This would slow down mass production due to its impracticality of extracting the camera from all the Wii Remotes. In addition, there is the risk that not all Wii Remote internals are from the same manufacturer, possibly resulting in inconsistent pinouts for the camera.

10 Project Recommendation of Implementation (System Integration)

The main objectives for the first senior design semester were met, we were able to have the main subsystems of the QUAD-S working properly i.e. sending accel/gyro sensor data into a simulation of our control system for analysis and correction using a PID controller. Although no real test was made for the control system, the simulations indicated that the control system should suffice and be ready for its real time implementation in the second part of senior design.

We were also able to get the sensing of IR LEDs with the Wii remote camera, which was able to detect the X and Y coordinates of four IR LED sources, along with their distances. We were also able to ensure that our infrared communication system was able to detect 4-bit pulse encoded ID values.

In the second part of Senior Design we will shift our focus onto the frame of the quadcopter, a circuit for speed control of the motors, the final PCB board design where all our components from the first part of senior design will be, and we will research about whether to go with brushed or brushless motors along with what propellers to choose.

11 References

[1] InvenSense Inc. (2012, May) InvenSense Innovation in MEMS. [Online]. http://www.invensense.com/mems/gyro/documents/PS-MPU-6000A.pdf

[2] Robot Electronics. [Online]. http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html

[3] InvenSense Inc. (2012, March) InvenSense Innovation in MEMS. [Online]. http://www.invensense.com/mems/gyro/documents/RM-MPU-6000A.pdf

[4] Jeff Rowberg. (2012) github. [Online]. https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/

[5] Joel Bartlett. MPU-6050_Breakout V11. [Online]. http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/IMU/MPU-6050_Breakout%20V11.pdf

[6] (2012, March) Wikipedia. [Online]. http://en.wikipedia.org/wiki/PID_controller

[7] (2009, June) Starlino Electronics. [Online]. http://www.starlino.com/quadcopter_acc_gyro.html

[8] Zhang, and Liu Jikai Peng. (2011) On New UAV Flight Control System Based on Kalman & PID. [Online]. http://ieeexplore.ieee.org/xpl/articleDetails.jsp;jsessionid= 12DqPLGRswqZJFJG4lFRnQr5j4VX3v1R1QLnmGW9BjZVyJbcWvJQ!399748823?arnumber= 6001516&contentType=Conference+Publications

[9] Vijay Kumar. (2012, March) TED: Ideas worth Spreading. [Online]. http://www.ted.com/talks/vijay_kumar_robots_that_fly_and_cooperate.html

[10] (2012, February) Wikipedia. [Online]. http://en.wikipedia.org/wiki/Kalman_filter

[11] CS4758. [Online]. http://www.cs.cornell.edu/Courses/cs4758/2012sp/materials/MI63slides.pdf

[12] Greg Czerniak. Greg Czerniak's Website. [Online]. http://greg.czerniak.info/node/5

[13] Bilgin Esme. Bilgin's Blog. [Online]. http://bilgin.esme.org/BitsBytes/KalmanFilterforDummies.aspx

[14] (2012, November) Wikipedia. [Online]. http://en.wikipedia.org/wiki/Alpha_beta_filter

[15] The PID Algorithm. [Online]. http://www.pidtuning.net/pid-algorithm.php

[16] Comparison of PID Control Algorithms. [Online]. http://www.expertune.com/artCE87.aspx

[17] William Etter. (2011, February) WilliamEtter.com Engineering. [Online]. http://williametter.com/portfolio/projects/wii-ir-camera-system/

[18] Kako. (2008, September) Kako Homepage. [Online]. http://www.kako.com/neta/2007-001/2007-001.html

[19] Peter "RobotFreak". (2009, June) Let's Make Robots! [Online]. http://letsmakerobots.com/node/7752

[20] Johnny Chung Lee. (2009, September) Procrastineering. [Online]. http://procrastineering.blogspot.com/2008/09/working-with-pixart-camera-directly.html

[21] Stephen Hobley. (2009, March) Pixart/Wiimote sensor library for Arduino. [Online]. http://www.stephenhobley.com/blog/2009/03/01/pixartwiimote-sensor-library-for-arduino/

[22] (2012, October) Wii Brew. [Online]. http://wiibrew.org/wiki/Wiimote

[23] (2012, May) Federal Aviation Administration. [Online]. http://www.faa.gov/news/updates/?newsId=68004

[24] (2012) Q&A Environmental Measures. [Online]. http://www.nintendo.co.jp/csr/en/q_and_a/qa5.html

[25] Mirage Palace. [Online]. http://maru-chang.com/hard/rvl/english.htm#RVL-001

[26] Stephany. (2012, November) Game Front. [Online]. http://www.gamefront.com/greenpeace-says-microsoft-and-nintendo-harmful-to-environment/

12 Appendices

Appendix-1. IR Tracking Main Program

```
// Example of using the PVision library for interaction with the Pixart
sensor on a WiiMote
// This work was derived from Kako's excellent Japanese website
// http://www.kako.com/neta/2007-001/2007-001.html

// Steve Hobley 2009 - www.stephenhobley.com
//=============================================================================
=================
// Modified by Angel Tijerina for the QUAD-S Senior Design I project
//
// Notes:
// Wii IR camera has 1024 x 768 resolution
// (0,0) is measured from Bottom-Left
// Camera in current configuration is "upside-down" - (0,0) is top right
// Center of camera is (1024/2, 768/2) = (512, 384)
// Information gathered from: http://wiibrew.org/wiki/Wiimote#IR_Camera

#include <Wire.h>
#include <PVision.h>
#include <math.h>

PVision ircam;
byte result;

//Blob tracking
int blobsFound;
Blob blobs[4];
Blob closestBlob;

void setup()
{
  Serial.begin(115200);
  ircam.init();
}

void loop()
{
  //Verify that the loop is working - for when the console is blank
  //Serial.print("pulse");

  //Camera object
  result = ircam.read();

  //keep track of how many Blobs were found, reset each loop
  blobsFound = 0;

  //Check for each BLOB
  if (result & BLOB1)
  {
    Serial.print("BLOB1 detected. X:");
    Serial.print(ircam.Blob1.X);
    Serial.print(" Y:");
    Serial.print(ircam.Blob1.Y);
    Serial.print(" Size:");
    Serial.print(ircam.Blob1.Size);

    //modified by Angel Tijerina to display distance
    Serial.print(" Distance:");
    Serial.println(ircam.Blob1.distance);

    //keep track of blob distance info
    blobs[blobsFound] = ircam.Blob1;
    blobsFound++;
  }

  if (result & BLOB2)
  {
    Serial.print("BLOB2 detected. X:");
    Serial.print(ircam.Blob2.X);
    Serial.print(" Y:");
    Serial.print(ircam.Blob2.Y);
    Serial.print(" Size:");
    Serial.print(ircam.Blob2.Size);

    //modified by Angel Tijerina to display distance
    Serial.print(" Distance:");
    Serial.println(ircam.Blob2.distance);

    //keep track of blob distance info
    blobs[blobsFound] = ircam.Blob2;
    blobsFound++;
  }

  if (result & BLOB3)
  {
    Serial.print("BLOB3 detected. X:");
    Serial.print(ircam.Blob3.X);
    Serial.print(" Y:");
    Serial.print(ircam.Blob3.Y);
    Serial.print(" Size:");
    Serial.print(ircam.Blob3.Size);

    //modified by Angel Tijerina to display distance
    Serial.print(" Distance:");
    Serial.println(ircam.Blob3.distance);

    //keep track of blob distance info
    blobs[blobsFound] = ircam.Blob3;
    blobsFound++;
  }

  if (result & BLOB4)
  {
    Serial.print("BLOB4 detected. X:");
    Serial.print(ircam.Blob4.X);
    Serial.print(" Y:");
    Serial.print(ircam.Blob4.Y);
    Serial.print(" Size:");
    Serial.print(ircam.Blob4.Size);

    //modified by Angel Tijerina to display distance
    Serial.print(" Distance:");
    Serial.println(ircam.Blob4.distance);

    //keep track of blob distance info
    blobs[blobsFound] = ircam.Blob4;
    blobsFound++;
  }

  //Serial.print("\n");

  //------------------------------------------------------------------
  // Examine information on blobs found
  Serial.print("Blobs found: ");
  Serial.println(blobsFound);

  // Loop through blobs array
  if(blobsFound != 0)
  {
    //assume first element smallest
    closestBlob = blobs[0];

    //check for other smaller distances
    for(int i = 1; i < blobsFound; i++)
    {
      if(blobs[i].distance < closestBlob.distance)
      {
        closestBlob = blobs[i];
      }
    }

    //print the closest blob
    Serial.print("Closest Distance: ");
    Serial.println(closestBlob.distance);

    //dictate action to take - Quadcopter is in "+" configuration
    //These
    /*
      /-----------------------------------------------\
      |                                               |
      |                      F                        |
      |                      |                        |
      |                  L --+-- R                    |
      |                      |                        |
      |                      B                        |
      |(0, 0)                                         |
      \-----------------------------------------------/
    */

    //check X - Left or Right
    if(closestBlob.X - ircam.c.X < 0)      //Left
    {
      Serial.println("Move Left");
    }
    if(closestBlob.X - ircam.c.X > 0)       //Right
    {
      Serial.println("Move Right");
    }

    //check Y - Forward or Reverse
    if(closestBlob.Y - ircam.c.Y < 0)       //Backward
    {
      Serial.println("Move Backward");
    }

    if(closestBlob.Y - ircam.c.Y > 0)       //Forward
    {
      Serial.println("Move Forward");
    }
  }

  //print a divider
  Serial.println("-----------------------------------------------------");

  //----------------------------------------------------------------
  // Short delay...
  //delay(100);
  delay(500);
}
```

Appendix-2. IR Tracking PVision Class Declaration

```
// PVision library for interaction with the Pixart sensor on a WiiMote
// This work was derived from Kako's excellent J
apanese website
// http://www.kako.com/neta/2007-001/2007-001.html

// Steve Hobley 2009 - www.stephenhobley.com

#ifndef PVision_h
#define PVision_h

#include "Arduino.h"
#include <Wire.h>
#include <math.h>

// bit flags for blobs
#define BLOB1 0x01
#define BLOB2 0x02
#define BLOB3 0x04
#define BLOB4 0x08


// Returned structure from a call to readSample()
struct Blob
{
      int X;
      int Y;
      int Size;
      byte number;
      double distance;
};

//added by Angel Tijerina to store the center of the Wii camera
struct Center
{
      int X;
      int Y;
};

class PVision
{

public:
      PVision();

      void init();   // returns true if the connection to the sensor
established correctly
      byte read();   // updated the blobs, and returns the number of blobs
detected

      // Make these public
      Blob Blob1;
      Blob Blob2;
      Blob Blob3;
      Blob Blob4;
```

```
	//Sensor Centor - added by Angel Tijerina
	Center c;

private:
	// per object data
	int IRsensorAddress;
	int IRslaveAddress;
	byte data_buf[16];
	int i;
	int s;

	void Write_2bytes(byte d1, byte d2);
	byte blobcount; // returns the number of blobs found - reads the sensor
};

// Arduino 0012 workaround
#undef int
#undef char
#undef long
#undef byte
#undef float
#undef abs
#undef round

#endif
```

Appendix-3. IR Tracking PVision Class Definitions

```
// Example of using the PVision library for interaction with the Pixart
sensor on a WiiMote
// This work was derived from Kako's excellent Japanese website
// http://www.kako.com/neta/2007-001/2007-001.html

// Steve Hobley 2009 - www.stephenhobley.com
/***********************************************************************
**
* Includes
************************************************************************
*/
#include "PVision.h"
#include <Wire.h>
#include <math.h>

/***********************************************************************
**
* Private methods
************************************************************************
*/
void PVision::Write_2bytes(byte d1, byte d2)
{
    Wire.beginTransmission(IRslaveAddress);
    Wire.write(d1); Wire.write(d2);
    Wire.endTransmission();
}


/***********************************************************************
**
* Constructor
************************************************************************
*/
PVision::PVision()
{
      Blob1.number = 1;
      Blob2.number = 2;
      Blob3.number = 3;
      Blob4.number = 4;

      //Initialize the center - added by Angel Tijerina
      c.X = 1024/2;
      c.Y = 768/2;

      /*
      Serial.print("Center X (");
      Serial.print(c.X);
      Serial.print(", ");
      Serial.print(c.Y);
      Serial.println(")");
      */
}

/***********************************************************************
**
```

```
* Public methods
*****************************************************************************
*/
// init the PVision sensor
void PVision::init ()
{
    IRsensorAddress = 0xB0;
    IRslaveAddress = IRsensorAddress >> 1;   // This results in 0x21 as the
address to pass to TWI

      //Send message to consol
      Serial.println("Initializing Sensor");

    Wire.begin();
    // IR sensor initialize
    Write_2bytes(0x30,0x01); delay(10);
    Write_2bytes(0x30,0x08); delay(10);
    Write_2bytes(0x06,0x90); delay(10);
    Write_2bytes(0x08,0xC0); delay(10);
    Write_2bytes(0x1A,0x40); delay(10);
    Write_2bytes(0x33,0x33); delay(10);
    delay(100);
}

byte PVision::read()
{
    //IR sensor read
    Wire.beginTransmission(IRslaveAddress);
    Wire.write(0x36);
    Wire.endTransmission();

    Wire.requestFrom(IRslaveAddress, 16);        // Request the 2 byte
heading (MSB comes first)
    for (i=0;i<16;i++)
    {
       data_buf[i]=0;
    }

    i=0;

    while(Wire.available() && i < 16)
    {
        data_buf[i] = Wire.read();
        i++;
    }

    blobcount = 0;

      //-------------------------------------------------------------------
    Blob1.X = data_buf[1];
    Blob1.Y = data_buf[2];
    s   = data_buf[3];
    Blob1.X += (s & 0x30) <<4;
    Blob1.Y += (s & 0xC0) <<2;
    Blob1.Size = (s & 0x0F);

      //modified by Angel Tijerina to calculate distance
      Blob1.distance = sqrt(pow((Blob1.X-c.X), 2) + pow((Blob1.Y-c.Y), 2));

    // At the moment we're using the size of the blob to determine if one is
detected, either X,Y, or size could be used.
    blobcount |= (Blob1.Size < 15)? BLOB1 : 0;

      //-----------------------------------------------------------------
    Blob2.X = data_buf[4];
    Blob2.Y = data_buf[5];
    s    = data_buf[6];
    Blob2.X += (s & 0x30) <<4;
    Blob2.Y += (s & 0xC0) <<2;
    Blob2.Size = (s & 0x0F);

      //modified by Angel Tijerina to calculate distance
      Blob2.distance = sqrt(pow((Blob2.X-c.X), 2) + pow((Blob2.Y-c.Y), 2));

    blobcount |= (Blob2.Size < 15)? BLOB2 : 0;

      //-----------------------------------------------------------------
    Blob3.X = data_buf[7];
    Blob3.Y = data_buf[8];
    s    = data_buf[9];
    Blob3.X += (s & 0x30) <<4;
    Blob3.Y += (s & 0xC0) <<2;
    Blob3.Size = (s & 0x0F);

      //modified by Angel Tijerina to calculate distance
      Blob3.distance = sqrt(pow((Blob3.X-c.X), 2) + pow((Blob3.Y-c.Y), 2));

    blobcount |= (Blob3.Size < 15)? BLOB3 : 0;

      //-----------------------------------------------------------------
    Blob4.X = data_buf[10];
    Blob4.Y = data_buf[11];
    s    = data_buf[12];
    Blob4.X += (s & 0x30) <<4;
    Blob4.Y += (s & 0xC0) <<2;
    Blob4.Size = (s & 0x0F);

      //modified by Angel Tijerina to calculate distance
      Blob4.distance = sqrt(pow((Blob4.X-c.X), 2) + pow((Blob4.Y-c.Y), 2));

    blobcount |= (Blob4.Size < 15)? BLOB4 : 0;

      //-----------------------------------------------------------------

    return blobcount;
}
```

Appendix-4. MATLAB script to plot the data from accel/gyro sensor

```
%%%%%%%%%
% QUAD-S
% Edson Gallegos
% Alex Marroquin
% Angel Tijerina
% Version 2
%Obtains the gyro data and plot its response
clc; clear; close all;

%serial setup
s = serial('COM5');          %Arduino on Serial COM5
s.BaudRate = 115200;
fopen(s);

%read opening lines
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)

%send start command
fprintf(s, 's')

a = fscanf(s)
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)

%keep track of time to plot
t = 0;
figure(1);


%Data Aquisition
for i = 1 : 5000

    %read next data
    a = fscanf(s);

    %parse data and print individual results in vector
    ypr = strread(a)

    %Separating the yaw to plot individually later
    y(i) = ypr(1);  %theta

    %Separating the pitch to plot individually later
    p(i) = ypr(2);  %psi

    %Separating the roll to plot individually later
    r(i) = ypr(3);  %phi
    %increment time
    t(i) = i;

end

%Plotting Yaw angles
figure(1)
plot(t/100,y)
grid on
title('Yaw');
xlabel('Time (sec)');
ylabel('Angle (deg)');

%Plotting Pitch angles
figure(2)
plot(t/100,p)
grid on
title('Pitch');
xlabel('Time (sec)');
ylabel('Angle (deg)');

%Plotting Roll angles
figure(3)
plot(t/100,r)
grid on
title('Roll');
xlabel('Time (sec)');
ylabel('Angle (deg)');
```

Appendix-5. SensorInput.m

```
%%%%%%%%%
% QUAD-S
% Edson Gallegos
% Alex Marroquin
% Angel Tijerina
%
% SensorInput.m
% Obtains the gyro data and processes it through the PID
% to obtain the necessary change in motor speed
clc; clear; close all;

%serial setup
s = serial('COM5');          %Arduino on Serial COM8
s.BaudRate = 115200;
fopen(s);

%read opening lines
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)

%send start command
fprintf(s, 's')

a = fscanf(s)
a = fscanf(s)
a = fscanf(s)
a = fscanf(s)

%constants & desired values
global past_err
past_err = 0;
theta_desired = 0;
phi_desired = 0;
psi_desired = 0;
g = 9.81;
m = 5;
K = 10;

%keep track of time to plot
t = 0;
figure(1);

%Initial values
a = fscanf(s);
ypr = strread(a)
y(1) = ypr(1);
p(1) = ypr(2);
r(1) = ypr(3);

w_hover = m*g/K;
```

```
wm1 = w_hover;
wm2 = wm1;
wm3 = wm1;
wm4 = wm1;

phi_prev = r(1);
psi_prev = p(1);

%Proper gains: Proportional, derivative, and integral in order
Kpsi = [0.017 0.0000002 0.000000002];
Kphi = [0.017 0.0000002 0.000000002];

%Data Aquisition
for i = 1 : 5000

    %read next data
    a = fscanf(s);

    %parse data and print individual results in vector
    ypr = strread(a)

    %Separating the yaw to plot individually later
    y(i) = ypr(1);  %theta

    %Separating the pitch to plot individually later
    p(i) = ypr(2);  %psi

    %Separating the roll to plot individually later
    r(i) = ypr(3);  %phi

    %increment time
    t(i) = i;

    %Calculate necessary speed change which is done by adding the control
    %signal to the hover speed required for the motor. If no change is
    %needed then the control will be 0 thus the motor speed will just be
    %the hover speed. A negative control will reduce the motor speed i.e.
    %the arm where the motor is located is above the horizontal plane
    %relative to the ground. A positive control will increase the motor
    %speed, which means the motor is below the same horizontal plane. The
    %motors work in tandem: one motor increases its speed while the
    %opposite motor decreases its speed.
    wm2(i) = Angle_Control(p(i), psi_desired, psi_prev, i, i-1, Kpsi) +
w_hover;
    wm4(i) = Angle_Control(-1*p(i), psi_desired, -1*psi_prev, i, i-1, Kpsi) +
w_hover;
    wm1(i) = Angle_Control(r(i), phi_desired, phi_prev, i, i-1, Kphi) +
w_hover;
    wm3(i) = Angle_Control(-1*r(i), phi_desired, -1*phi_prev, i, i-1, Kphi) +
w_hover;

    %Store previous angle values
    phi_prev = r(i);
    psi_prev = p(i);
```

```
end

%Plotting Yaw angles
figure(1)
plot(t,y)
grid on
title('Yaw');
xlabel('Time (sec)');
ylabel('Angle (deg)');

%Plotting Pitch angles
figure(2)
plot(t,p)
grid on
title('Pitch');
xlabel('Time (sec)');
ylabel('Angle (deg)');

%Plotting Roll angles
figure(3)
plot(t,r)
grid on
title('Roll');
xlabel('Time (sec)');
ylabel('Angle (deg)');

%Motor 1 Speed
figure(4)
plot(t,wm1)
grid on
title('Motor 1 Speed');
xlabel('Time (sec)');
ylabel('Speed (rad/sec)');

%Motor 3 Speed
figure(5)
plot(t,wm3)
grid on
title('Motor 3 Speed');
xlabel('Time (sec)');
ylabel('Speed (rad/sec)');

%Motor 2 Speed
figure(6)
plot(t,wm2)
grid on
title('Motor 2 Speed');
xlabel('Time (sec)');
ylabel('Speed (rad/sec)');

%Motor 4 Speed
figure(7)
plot(t,wm4)
grid on
```

```
title('Motor 4 Speed');
xlabel('Time (sec)');
ylabel('Speed (rad/sec)');
%%%%%%%%
```

Appendix-6. AngleControl.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [w_motor] = Angle_Control(theta_actual, theta_desired, theta_prev,
t1, t0, K)
% AngleControl.m
% Receives as inputs current angle, desired angle, 2 time points and
% previous motor speed

m = 5;       %Mass
Kl = 10;     %Transfer function from thrust to motor speed. Assumed to be a
constant.
radius = 1; %Arm length
control = PID(theta_actual, theta_desired, theta_prev, t1, t0, K, 1);%
Corrective factor

% Since the control output is the required change in angle the angular
% velocity can be calculated by dividing it with respect to 2 time points,
% t1 and t0. The acceleration is simply the angular velocity divided by the
% 2 time points.
wc = control/(t1 -t0);
ang_a = wc/(t1 -t0);
tang_a = radius*ang_a;

% Thrust as a function of motor speed, T = f(w_motor) = K*w_motor. To find
% the motor speed we simply take the inverse of the function on thrust,
% w_motor =  f^-1(T) = T/K. Thrust is a function of tangential acceleration
% and mass, T = m*tang_a = m*radius*ang_a.


% The motor speed is changed only when speed/orientation needs to be
% changed.  When desired orientation is achieved the motor speed should
% remain the same i.e. the output should be zero.
w_motor = m*tang_a/Kl;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Appendix-7. PID.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [e] = PID(actual, desired, previous, t1, t0, K, which)

global past_err

% Numerical integration
past_err = past_err + actual*(t1 - t0);
% Difference between desired and actual values
u = desired - actual;
% Numerical derivative
ud = (actual - desired)/(t1-t0) - (actual - previous)/(t1-t0);
% Difference between the area under the curve if stable from 1st time
% sample and the actual area under the curve
uI = desired*t1 - past_err;

% The which variable determines whether the Parallel or Series PID is
% implemented. For testing purposes only.
if(which == 0)
    %Parallel PID
    Ep = K(1)*u;
    Ed = K(2)*ud;
    Ei = K(3)*uI;
    e = Ei + Ed + Ep;
else
    %Series Implementation
    Ed = K(2)*ud + 1;
    Ei = K(3)*uI + u;
    e = K(1)*Ed*Ei;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```