FRESNO STATE
Lyles College of Engineering

MEMORANDUM (Reece)

Date: 18 May 2017

To: Dr. Gregory Kriehn
Dr. Ram Nunna

From: Khai Chang
Reece Mulhern
Rahul Nunna

Subject: GPS-Denied Formation Control Methodologies in Unmanned Aerial Systems - Final Report

The purpose of this document is to provide a summary of the GPS-Denied Formation Control of Unmanned Aerial Systems. A review of the problem and the solution will be presented with the supporting results from the individual subsystem and the integration to complete the system. The methodologies, implementations, difficulties, and successes will be discussed. The document provides reasoning on the implementations and of the deviations from the original designs. The project promised deliverables will be discussed in the scope of the completed deliverables. A summary of the resources to complete the project and project time line will be explained.

GPS-Denied Formation Control Methodologies in Unmanned Aerial Systems

ECE 186B - SENIOR DESIGN II - Attack of the Drones

Spring 2017

18 May 2017

Khai Chang

Reece Mulhern

Rahul Nunna

Instructor and Technical Advisor: Dr. Gregory Kriehn

Signature

Date

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/DEFINITIONS

API - Application Program Interface

CAD - Computer Aided Design

ESC - Electronic Speed Controller

FAA - Federal Aviation Administration

FCC - Federal Communication Commission

GPS - Global Positioning System

I2C - Inter-Integrated Circuit

IEEE - Institute of Electrical and Electronics Engineers

IMU - Inertial Measurement Unit

I/O - Input Output

LED - Light Emitting Diode

NB - Narrow Band

PCB - Printed Circuit Board

PPM - Pulse Position Modulation

PWM - Pulse Width Modulation

RTL - Return to Launch

SPI - Serial Peripheral Interface

TCP/IP - Transmission Control Protocol Over Internet Protocol

UART - Universal Asynchronous Receiver/Transmitter

UAS - Unmanned Aerial System(s)

UDP/IP - User Datagram Protocol Over Internet Protocol

USB - Universal Serial Bus

USRT - Unmanned System Research Team

UWB - Ultra-Wideband

1. EXECUTIVE SUMMARY (REECE)

This document aims to report on the development, testing and implementation of the embedded system for the control of autonomous aerial vehicle formation, specifically the application of a leader follower formation control. With the recent significant body of work done in the field of multi-robot formation control, much of this research is reliant on the use the Global Positioning System (GPS) for absolute localization in the global reference frame. However, GPS is not a feasible localization methodology in all environments. This is due to GPS's reliance on multiple satellites as well as its limited resolution ($\approx$3m). GPS can also often at times provide inconsistent or fault data in indoor environments. As a result, there is a need for a a localization methodology capable of working with robotic control and formations in smaller enclosed environments.

The goal of this work was to develop a GPS denied formation control system via a two step approach. The first step was to establish the localization of each individual Unmanned Aerial System (UAS) via ultra-wideband radio (UWB) localization. By using time of arrival data between each UAS and a set of external static UWB modules, the location of each UAS can be found using the concept of bilateration for two dimensional and trilateration for three dimensions. The second step of the formation control was to apply the application of this localization date into a leader follower altitude matching algorithm to be integrated into the platform. The implementation of a closed loop control algorithm on an embedded Linux processor was to create the formation control system. On each UAS with an integrated PixHawk module (running Ardupilot autopilot platform), the embedded system pulls the distance data from the UWB module to calculate the location of the UAS. The manually controlled leader UAS, transmits it's location to the follower. The follower system controller processes the leader location and determines the travel distance required to match the leader. The communication transmission between the UASs, occurs through an ad hoc Wi-Fi network. This solution eliminates the need for any external infrastructure (such as GPS), and the abstraction of control to a separate processor allows for the system to be integrated into any autopilot system running the Ardupilot platform.

This project had three main deliverables. The first was the development of all the necessary subcomponents (embedded Linux computer, communication system, UWB radio) into a standalone system capable of interfacing with the Pixhawk system. The second was the development of a localization algorithm for each UAS, using bilateration between UWB radios. The third was the development of a leader-following algorithm, using the determined locations of two UAS to control the follower to match the altitude of the leader.

2. PROBLEM STATEMENT

2.1. General Problem Statement

The aim of this project is to develop a leader-following control methodology for a group of multi-rotor UAS to autonomously match the altitude of a manually-controlled leader UAS. Each of the ”follower” UAS must not be manually controlled by a human operator, and instead must only respond to changes in the leader's altitude. In order to accurately coordinate the motion of such a formation of independent UAS, it is necessary to localize each unit in a three-dimensional space. Much of the research done in this field employs the use of GPS for localization. However, there are few localization solutions available in the absence of GPS. In addition, there is no current standalone formation control solution available that can directly interface with the ArduPilot platform. All current solutions require direct changes to the ArduPilot firmware or a standalone third-party autopilot firmware with formation control capabilities built in. This project aims to develop a standalone formation control system that can directly integrate into the control schema of the ArduPilot platform.

2.2. General Solution Statement

Current research into the determination of physical location and spatial data of objects in space while being deprived of GPS data is of great interest for a wide variety of applications. The goal of this project is to design and implement a localized spatial location determination system within a multi-rotor copter swarm. The requirements of the system will be the implementation of a minimum of two copters, one manually operated, and the second receiving its leader's altitude data, and autonomously matching its altitude. Each UAS will be localized using trilateration via ultra-wideband radio. By using UWB technology, each UAS can be localized to a precise resolution, and can then use that localization data to produce accurate formations.

3. CURRENT STATE OF THE ART

Much work has been done in the fields of automated control of UAS as well as distributed control of mobile robots, as well as the combination of both research areas. Several attempts have been explored in recent years to pursue distributed control of multi-agent robotic networks, using constrained formations [1], [2]. Much of this research initially was developed for ground vehicle formations, however recently several forays have been made into the application of these algorithms to aerial vehicles [3]. Aerial vehicles have their own challenges, including (but not

limited to): additional dimensions of motion, lack of precise localization through the use of shaft encoders, and the general reduction of stability in aerial vehicles as opposed to ground vehicles. Many researchers have mitigated the issue of localization by resorting to triangulation/trilateration practices, with the use of GPS or UWB radio modules [4]. However, the limitation with systems such as these are their reliance on external infrastructure, either through the use of GPS satellites or externally placed UWB radios.

4. PIXHAWK 2 WITH DJI F550 HEXACOPTER (KHAI)

The Pixhawk 2 flight controller is the latest development from 3DR Robotics and its open source community to allow pilots to design and customize flights. For the design of the project, the flight controller plays the role as the main component for operating the UAS since it engages pilots to organize scenarios of how flight operations are conducted under different situations. In this case, the Pixhawk 2 is used to operate the DJI F550 hexacopter to perform guided waypoint flying in real time. Flight data is then captured using a companion computer that is integrated with flight controller to transmit data to neighboring UAS. This section specifically covers the structure and configuration of the Pixhawk 2 on the hexacopter, given that the UAS is already assembled.

Before assembling the components of the flight controller, it is essential that a GPS module, telemetry radio, and Pulse Position Modulation (PPM) encoder are inlcuded when connecting to Pixhawk 2. The PPM encoder, however, is not required when assembling the system, and is only necessary if the receiver of the Radio Control (RC) does not already have a built-in PPM encoder. The purpose of the encoder is to translate up to eight Pulse Width Modulation (PWM) signals into one PPM signal for the flight controller to read. Of course, both GPS module and telemetry radio are required to triangulate location of UAS and to communicate with the ground station.

By examining the diagram in 4.1, the system represents the top-level design of the Pixhawk 2 flight controller with the hexacopter. As shown, an 11.1 Volt lithium polymer (LiPo) battery is used to supply power throughout the system. Since the Pixhawk 2 cannot support such high voltages, the battery is connected to a power module to reduce the voltage to five volts. The power module can be found within the flight controller kit and assists in lowering the voltage while supplying sufficient power to each Electronic Speed Controllers (ESC). In addition, each speed controller manages the speed and direction of one electric motor. To accomplish this, speed controllers are connected directly to separate channels on the Pixhawk 2 and communicates

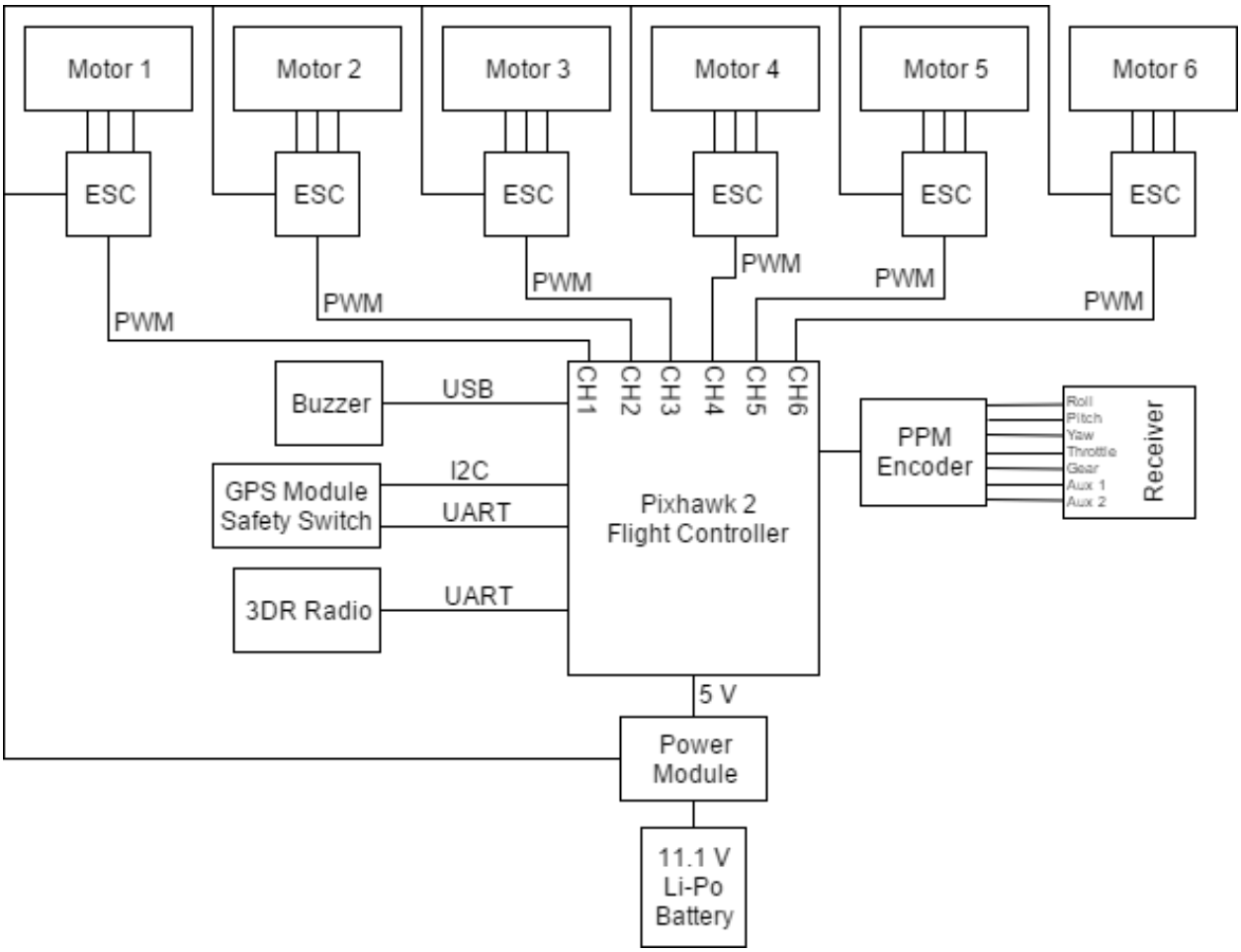


Figure 4.1: Top Level System Design of Pixhawk 2 and Hexacopter

using PWM. As such, channel one is connected to motor one's speed controller, channel two to motor two's speed controller, and so on until all speed controllers have been connected. Other components such as buzzers, telemetry radios, and GPS modules are then connected to the controller and communicates through either Unverisal Asynchronous Receiver/Transmitter (UART) or Inter-Integrated Circuit (I2C) communication protocols.

4.1. Pulse Position Modulation (PPM) Encoder

For this design, the PPM encoder is required to bridge the receiver to the flight controller since the DEVO 7 receiver does not implement PPM encoding to communicate directly to the Radio Control (RC) input channel. Since the encoder has already been preprogrammed to read specific channels as either throttle, pitch, roll, or yaw, PWM signals from the receiver must be connected

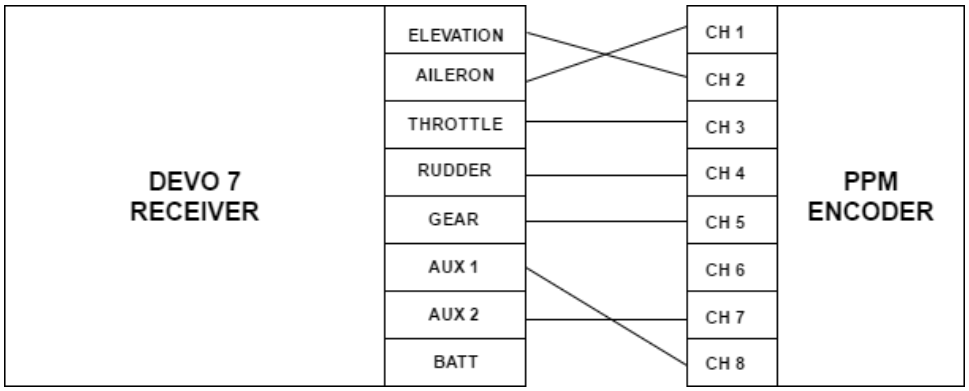


Figure 4.2: Receiver - PPM Encoder Channel Assignment

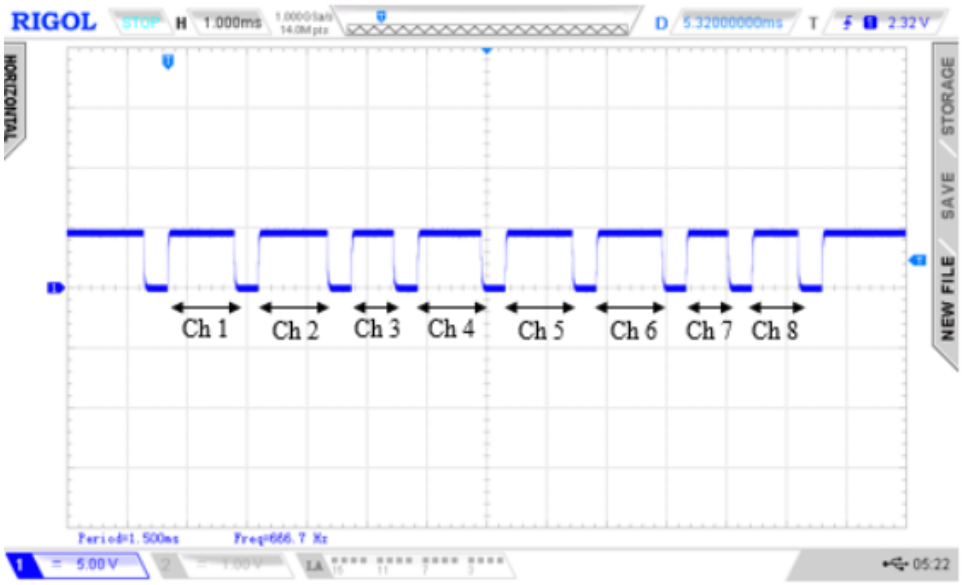


Figure 4.3: PPM Waveform

to the correct channel in order to ensure that actions of the UAS matches the pilot controls. Other channels of the encoder are then programmed to read inputs such as toggle switches and knobs from the transmitter. These inputs can then be configured to switch to different modes and perform emergency maneuvers if necessary.

As shown in 4.2, connections of channels one through four has been configured to read pitch, roll, yaw, and throttle signals coming from the receiver. Channel ones connection to aileron represents the roll signal from the transmitter to allow the UAS to lean to the left or right based on the pilots controls. To move the UAS forward and back, channel twos connection to elevation represents the pitch signal. Throttle is connected directly with channel three, and controls the

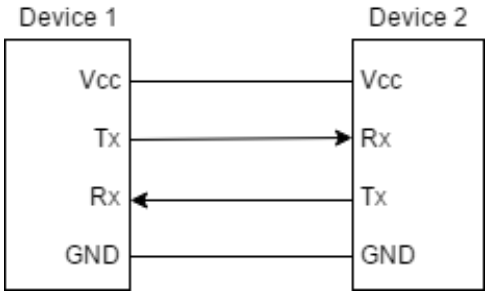


Figure 4.4: UART Connections

velocity of the motors to change to desired altitude. Lastly, channels four connection to rudder controls the yaw of the aircraft to change the orientation of where it is facing. Other channels, such as five through eight, are configured to read toggle switches from the transmitter. However, channel five is exclusively used to set the different modes when flying, and can be configured within the program.

When examining the PPM signal in 4.3, the total length of the frame is roughly 18 to 20 milliseconds and contains eight channels. As stated previously, each channel represents the inputs from the transmitter and can vary in duration. In between each channel is the signal low state, and is estimated to be 0.3 milliseconds in order to differentiate between the different pulses of the channels. In its ground state, the PPM signal to the UAS is as shown above. When arming and raising the throttle of the copter, channel three of the PPM signal increases the duration of its high state for the Pixhawk 2 to decode and distribute the PWM signals to each speed controllers.

4.2. Universal Asynchronous Receiver/Transmitter (UART) Communication Protocol

The Universal Asynchronous Receiver/Transmitter (UART) communication protocol is a form of serial communication used to send and receive data between devices. Most significantly, UART does not require a clock signal when transmitting and receiving data as compared to other communication protocols such as Serial Peripheral Interface (SPI) or I2C. To accommodate the lack of a clock signal during the process, UART allows users to specify the timing information of the hardware by defining the baud rate in which the devices are communicating in bits per second. Unsurprisingly, UART communication doesnt work if the transmitter and receiver have been configured for different data-transmission frequencies [5].

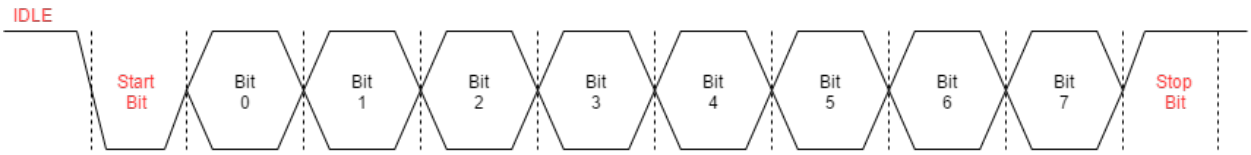


Figure 4.5: UART Data Frame

In the diagram above, 4.4 demonstrates the connections and directions of data transmission between device 1 and device 2. As such, both devices share a common power source and ground in exception to both Tx and Rx. In addition, Tx represents the transmitting signal while Rx is receiving signal. When sending information from device 1, data is sent via the Tx port and received by device 2 on the Rx port, and vice versa for forwarding information from device 2 to device 1.

Furthermore, by closely examining the data frame in 4.5, the system is typically in an idle state where no data is transmitted or received. When a start bit is indicated, the data line leaves its idle state and begins to send meaningful data before reaching its stop bit. When the stop bit is indicated, transmissions stops and returns to its idle state. Since both start and stop bits are overhead bits, it only functions to facilitate communication but does not transfer meaningful data [5].

The UART communication protocol is significantly important for the design of the project since it is the main method of interfacing between multiple components within the system during the integration process.

4.3. Mounting Pixhawk 2 to UAS

Before mounting the Pixhawk 2 flight controller onto the DJI F550 Hexacopter, motors must be placed in the correct order so that the UAS does not spin out of control. Motors are mounted where its adjacent motor is spinning in the opposite direction. For instance, if motor one spins counter-clockwise, its adjacent motors must spin in the clockwise direction. To help simplify this issue, all motors are clearly labeled with the direction of which they are designed to turn.

The Pixhawk 2 is then mounted as close as possible to the center of gravity of the hexacopter in both horizontal and vertical directions. In addition, the controller must face the same direction as the front of the copter as indicated by the arrow on top of the module. This is a standard method of mounting the Pixhawk 2 and is highly recommended by Ardupilot. However, it is not

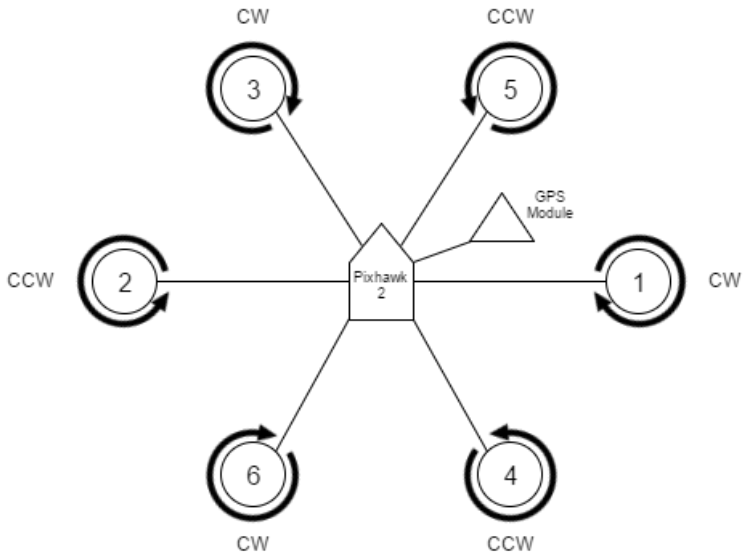


Figure 4.6: Pixhawk 2 Motor Order with GPS Module

the only location for mounting since the device is capable of being place elsewhere, so long as the position is configured within Mission Planner.

In continuation, 4.6 demonstrates the essential system level requirements for the unmanned vehicle. This includes mounting the GPS module on the outside of the frame and facing the same direction as the Pixhawk 2. For better chances of GPS lock, the module is mounted on an elevated position with a clear view of the sky. When powered on, a solid red Light Emitting Diode (LED) and flashing blue LED indicates that GPS lock has been acquired. Besides the GPS module, motor connections are shown as in the figure where channel one's connection is to the far right motor. Each number represents the motor order in which the speed controllers are connected to the channels on the main controller.

5. MISSION PLANNER (KHAI)

After the integration of the Pixhawk 2 and hexacopter, the next step in the process requires the configuration of the multirotor before initialy flying. To accomplish this, the Ardupilot community offers the Mission Planner program to interface with the Pixhawk 2 to customize and configure flight operations. The latest version of Mission Planner can be downloaded and installed from the Ardupilot website for Windows 7 operating systems or higher. For Mac and Linux users, Ardupilot also offers APM Planner which functions simiarly to to that of Mission Planner.

5.1. First Time Configurations

For first time configurations, the latest firmware needs to be flashed onto the board for the controller to recognize the system is setup for a hexacopter. This can be done by connecting the Pixhawk 2 to a computer using the Unverisal Serial Bus (USB) to micro-USB cable. At this time, Mission Planner is opened to detect that the device is connected and accessible. Users can then select the communication port on the program and connect to the device at a baud rate of 115200 bits per second. This particular baud rate is only used if connecting directly to the flight controller from computer. Else, the baud rate is adjusted 57600 bits per second when communicating over telemetry. However, when first flashing the firmware, it is important that users do not connect to the Pixhawk 2 just yet.

Under the 'INITIAL SETUP' tab, pilots can select 'Install Firmware' to flash the latest firmware for the hexacopter design. During that process, physical connection from the Pixhawk to the computer must not be disconnected at any point. Once upload is complete, a musical tone from the buzzer will sound to indicate that the firmware was successfully updated. If program failed to install the firmware, a different tone will sound to indicate failure. All of the sounds from the buzzer can be found on the Ardupilot website documentation to better understand the status of the copter. When prompted, power cycle the Pixhawk 2 to apply changes.

Once firmware has been installed, users can connect to the device communication port to finish configurations. Under the 'CONFIG/TUNING' tab, users can view complete parameters of the device by selecting 'Full Parameter List'. From the list, the 'AHRS ORIENTATION' parameter describes the board orientation on the copter and needs to be set to 0. By default, the parameter value is set to 24, which describes the flight controller to be pitched 90 degrees on the copter. Thus, the reason why the artificial horizon is upside-down. Finally, click on the 'Write Param' button to apply changes.

In addition, due to a hardware design flaw for Pixhawk 2 modules that were produced before 2017, the channel 1 port is shorted and is not recommended when flying. To accommodate for the design flaw, simply move motor 1's PWM connection to channel 7. In the parameter list, set 'RC7 FUNCTION' value to 33 to indicate that channel 7 is no operating as motor 1. Lastly, write parameters to device and power cycle flight controller to apply and view changes.

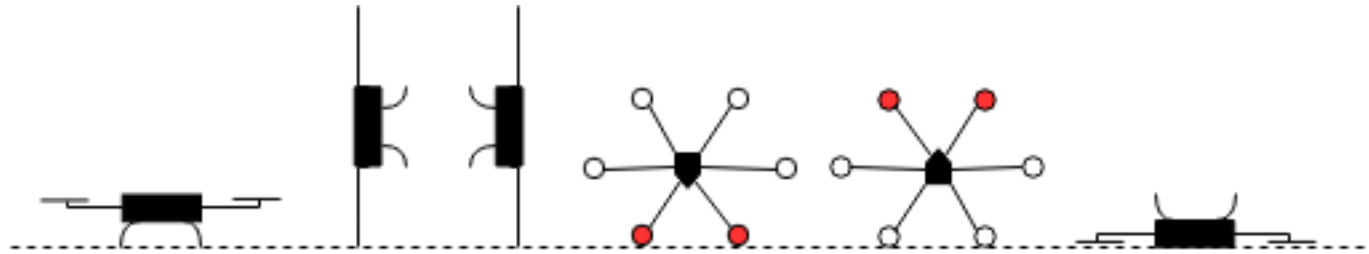


Figure 5.7: Accelerometer Calibration

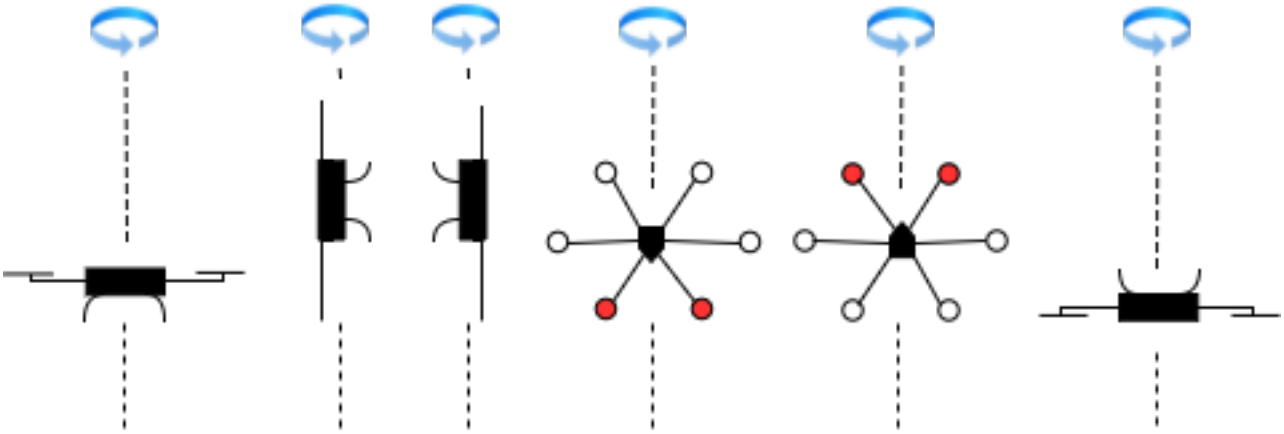


Figure 5.8: Compass Calibration

5.2. Calibrating UAS for Flying

When using the 3DR telemetry radio, this allows the vehicle to communicate with the ground staion wirelessly. This is especially useful when calibrating the vehicle to the area where it is flying. To do so, connect one 3DR telemetry radio to the computer via USB to micro-USB cable and another radio to the telemetry port of the Pixhawk 2. Transmitter can then be turned on before powering on the hexacopter with LiPo battery. On the computer, a new device communication device port is detected and be connected to with a baud rate of 57600 bits per second.

Similar to the previous section, the program wizard to perform all calibrations can be found under the 'INITIAL SETUP' tab. When prompted, the vehicle and frame type desired is a hexacopter frame with a QUAD X layout. Moving forward, to calibrate acclerometers, pilots must place vehicle on all of its axis as shown in 5.7. It is important to note that copter must be on ground level of where is planning to takeoff and land since it defines the altitude of the vehicle in respect to the area's ground level. Next, to calibrate the compass of the vehicle, pilots

Flight Mode	Description
Stabilize	• Manual flight mode. • Automatically levels and maintains current headway. • Pilot has full control.
Loiter	• Copter maintains consistent location, heading, and altitude. • Pilot has full control. • Slow to a stop and hold position when joysticks are released. • Requires GPS lock.
Return-to-Launch (RTL)	• Navigates copter from its current position to hover above home location. • Rise to 15 meters before returning home. • Home location is where it is first armed. • Requires GPS lock.
Auto	• Copter will follow pre-programmed mission script. • Roll, pitch, and throttle inputs are ignored. • Autopilot retakes yaw control as vehicle passes next waypoint. • Requires RTL or Land command to end mission. • Requires GPS lock.
Guided	• Guide copter to target location wirelessly using telemetry radio modules and ground station. • Requires 3DR radio and Mission Planner / Mavproxy. • Set waypoints by clicking on map in program. • Requires GPS lock.
Land	• Attempts to bring copter straight down. • Upon reaching ground, copter will automatically disarm. • Requires GPS lock.

Figure 5.9: Flight Modes

must rotate the copter 360 degrees in the air on each of its sides as shown in 5.8. For better results, rotating the unmanned system more than 360 degrees allows the program to collect more data samples.

Lastly, radio control calibration is needed in order to operate the vehicles movement and orientation. To accomplish this, joysticks and toggle switches on the transmitter are moved to their maximum positions to determine what are the maximum and minimum PWM range of each input. It is also important to note that this calibration is done without any propellers on the motors due to the fact that motors may spin when calibrating throttle stick. To avoid situations such as that, propellers are removed until after confirming that copter is safe to operate. Once radio controls have been calibrated, channel five can configured to operate different modes. As shown in 5.9, stabilize mode allows the pilot to fully control the copter without relying on GPS. On the other hand, loiter mode operates similarly to stabilize mode, but with a GPS lock to maintain location, heading, and altitude. Unfortunately, the DEVO 7 transmitter is only capable of recognizing two modes in a single toggle switch. Based on the goals of the design, loiter mode and guided mode were higher priority since it offers a more safe and controlled testing

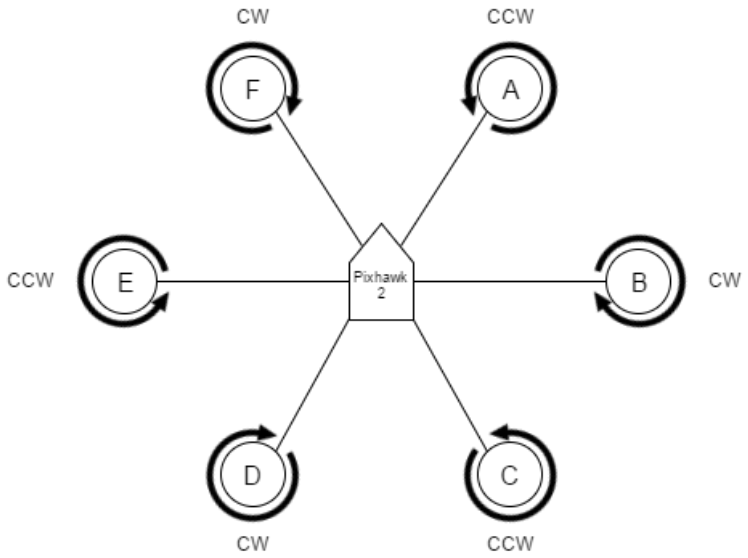


Figure 5.10: Motor Direction

environment when executing flight operations. Of course, RTL and land mode are necessary since they act as the failsafe in case of emergencies, and can be programmed to operate a single switch through channels 7 and 8.

Additionally, to improve chances of a safer flight operation, a motor test is usually performed to determine whether individual motors are spinning in the correct directions. Motors can be tested by navigating to 'Optional Hardware' section under the 'INITIAL SETUP' page. For a hexacopter frame, motor directions can be referenced to 5.10. If a motor is spinning in the wrong direction, switch two of three connections from the electric motor to the speed controllers to reverse the direction. If motors are spinning in the proper direction, no changes are required afterward.

5.3. Mission Planning

As a result, the copter is ready to take its first flight to determine whether anymore configurations are needed on Mission Planner. From this point forward, a controlled test is needed to verify that inputs from transmitter are correct, and that emergency switches are operational. Thus, to verify controls of transmitter, the pilot arms the motor and immediately disarm to verify controls on the ground. In continuation, the hexacopter is set to loiter mode to verify that throttle is working properly by slowly raising its input. If the UAS has no issue taking off, then the next step is to verify pitch, roll, and yaw controls.

Figure 5.11: Controlled Test Flight

In 5.11, the figure demonstrates the flight path taken to verify pitch, roll, and RTL while in loiter mode. Based on the figure, the hexacopter was first armed in the top right corner, as indicated by the home point. When pitched forward and rolled left, the copter moves forward and to its left while maintaining its heading. When rolled right, the copter moved directly to its right before coming to a stop. In its final phase, it rolls back left and pitched forward again. RTL is then triggered at the very end to fly directly back and hover above home point.

6. COMPANION COMPUTER WITH DRONEKIT (KHAI)

Due to the fact that the Pixhawk 2 flight controller has a direct connection for the Intel Edison module, real time data of flights operations are shared between both modules. With dronekit, it is possible to program the Intel Edison as a companion computer by scripting python codes. For the purposes of this design, the Intel Edison is used to extract data such as velocity and orientation from UAS while interfacing with other components to determine localization.

6.1. Dronekit-Python

DroneKit-Pythons purpose is to allow developers to create applications that can run on board a companion computer. In this case, its applications are set to run on the Intel Edison module

in the Pixhawk 2. This is because Dronekits API allows communication with the UAS via MAVLink protocols, similar to how Ardupilot communicates to the ground station through the same protocol. Thus, since of every modules communication through MAVLink, telemetry data is shared amongst the companion computer, Pixhawk 2, and ground station.

6.2. Raspbery Pi Zero Wireless

Over the progression of the project, the Intel Edison module that connects directly to the Pixhawk 2 does not supply voltage over its serial port. Overall, the issue poses a huge blockage in the progression of the project when interfacing with the UWB modules. Fortunately, the Raspberry Pi Zero Wireless is supported by Pixhawk 2 and operates as a companion computer as well. Benefits of the Raspberry Pi Zero Wireless over the Intel Edison module is its support from the community since the Edison is no longer supported by Intel. Disadvantages of the Raspberry Pi is the processing power compared to the Intel Edison module.

7. XBEE S2C RF MODULE (KHAI)

The purpose of the XBee modules are to transmit and receive data from leader UAS to follower. Its design and characteristics allows for low power consumption while communicating up to distances as long 1200 meters line-of-sight outdoors. When communicating between XBee modules, at least one module must be coordinator. As coordinator, it is capable of transmitting data to routers or end devices. Xbee modules that are assigned as routers can then transmit data to other routers or end devices around them. However, the use of XBee S2C modules for communication is inefficient since the rate of transmission is far too slow for sending real-time position data from leader to follower.

8. USER DATAGRAM PROTOCOL (UDP) (KHAI)

User Datagram Protocol (UDP) is a form of internet communication protocol in which it sends messages in the form of datagrams to other hosts on the Internet Protocol (IP) network. With the use of UDP, the focus is towards just sending packets on a much lower bandwidth overhead and latency. Disadvantages of UDP versus other methods of network protocol, such as Transmission Control Protocol (TCP), is the possibility of missed packets on the receiving end. TCP/IP is a secured method of network communication, but usually contains a greater overhead and latency due to system always checking for lost packets. Ideally, using UDP/IP is more efficient for this

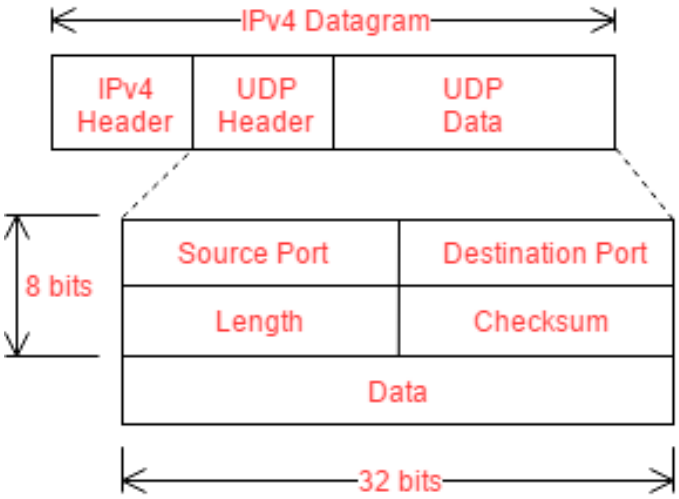


Figure 8.12: UDP/IP Datagram

project since location data needs to be constantly streaming from leader to follower. Missed packets may be present but does not impede the follower from calculating its distance from leader.

8.12 demonstrates the structure of the UDP/IP datagram. As shown, the IPv4 header defines the destination address in which the packet is being sent to. After the IPv4 header is the UDP datagram that consists of another header and data that is 32 bits wide. Within the overall structure of the project, the leader creates a UDP socket datagram and specifies the IP address of follower as the destination. Additionally, both devices must be operating under the same port in order for information to be acquired. In this case, the leader is continuously streaming its location data while the follower is constantly listening for incoming data.

9. TRILATERATION AND UWB CONFIGURATION (REECE)

To implement the leader follower control system, the UAS need a method to determine their individual relative locations from a defined coordinate system. With the design criterion to implement the system without the aid of the GPS, a local coordinate system is being defined by an anchor tag UWB radio modules. The choice of UWB radio modules was made due to UWB signals being less effected by noise and multi-path interference due to the high frequency content and the sharpness of the pulse signals.

Within the anchor tag system, the anchors act as coordinate constants that define the system, while tags act as moving points in the local coordinate system. This was done by implementing a trilateration system to define the coordinate system. Trilateration is a method of spatial geometry

using the intersections of spheres to define coordinate systems and discover locations within the coordinate systems. Trilateration requires known distance between the centers of spheres, viewing the intersections of said spheres, and measuring the radii at the intersections. Due to the geometry of the spheres the intersection points result in a localization. The proposed system of one dimensional matching between the two hexacopters requires two anchors to define the system and a tag each for the two UAS, meaning that there is a circle defined on a plane that is perpendicular to the direct line between the two centers of the spheres. The Trek 1000 kit produced by Decawave was chosen due to it containing four UWB modules that can be configured as either anchors or tags. For this projects purposes two were configured as tags and two as anchors.

9.1. UAS Unit Localization (Reece)

Each UAS was localized in an enclosed three-dimensional space through the use of UWB radio trilateration. UWB radios, placed to determine the local reference frame, measured their distance to each UAS in a given formation. The distance from each UWB satellite r to a given UAS can be described by a sphere with radius r.

In multi-dimensional geometry, it can be shown that to determine the position of an object in an n-dimensional space, the number of spheres intersecting required to determine the locations relative to the centers of said spheres is equal to n+1 [4]. Investigating three dimensional space, intersecting two spheres there exists an circle that is a constant radial distance from the center of the two spheres. By applying a third sphere, another circle becomes defined. Between the intersection of the two circles, two points in space become defined that are constant radial distances between the three spheres. By repeating this process with a fourth sphere, the two remaining intersections are narrowed down to a single point that is radial from all four spheres. The following derivation illustrates the application of the intersection of the spheres to determine the one dimensional location of the rotor copters.

Known: a

Measured: r_1, r_2, r_p

$$r_1^2 = x^2 + y^2 + z^2 \quad \text{(Eqn 9.1)}$$

$$r_2^2 = x^2 + y^2 + (z-a)^2$$

$$= x^2 + y^2 + z^2 - 2az + a^2 \quad \text{(Eqn 9.2)}$$

- Subtract (Eqn 9.2) from (Eqn 9.1)

$$r_1^2 - r_2^2 = x^2 + y^2 + z^2 - x^2 - y^2 - z^2 + 2az - a^2$$

$$r_1^2 - r_2^2 = 2az - a^2 \quad \text{(Eqn 9.3)}$$

- Solve for Z

$$r_1^2 - r_2^2 + a^2 = 2az$$

$$z = \frac{r_1^2 - r_2^2 + a^2}{2a} \quad \text{(Eqn 9.4)}$$

- This results in a plain at this height of 'Z'.

- Substitute this result into (Eqn 9.1) to solve for sphere intersection.

$$r_1^2 = x^2 + y^2 + (\frac{r_1^2 - r_2^2 + a^2}{2a})^2$$

- Collect constants and define radius of the intersection circle.

$$r_1^2 - (\frac{r_1^2 - r_2^2 + a^2}{2a})^2 = x^2 + y^2 \quad \text{(Eqn 9.5)}$$

$$r_p = \sqrt{r_1^2 - (\frac{r_1^2 - r_2^2 + a^2}{2a})^2} \quad \text{(Eqn 9.6)}$$

- Substituting (Eqn 9.6) into (Eqn 9.5)

$$r_p^2 = x^2 + y^2 \quad \text{(Eqn 9.7)}$$

To generate these spheres, off board UWB radio modules will be used to define a relative coordinate system. Equations EqnEqn 9.1 and EqnEqn 9.2 will be used to define to spheres in 3 dimensional space. The sphere from EqnEqn 9.1 define a coordinate system with its centroid at Cartesian coordinates (0,0,0) and the sphere from EqnEqn 9.2 being shifted in the positive Z-direction. Initial configurations will follow with the ground being the XY-plane and

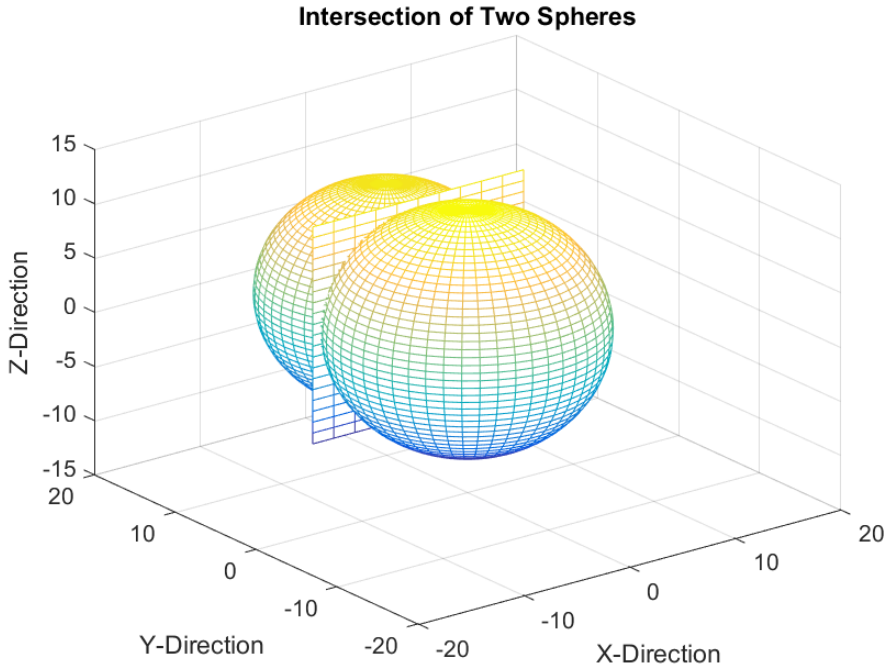


Figure 9.13: Intersection of Two Spheres - 3D View.

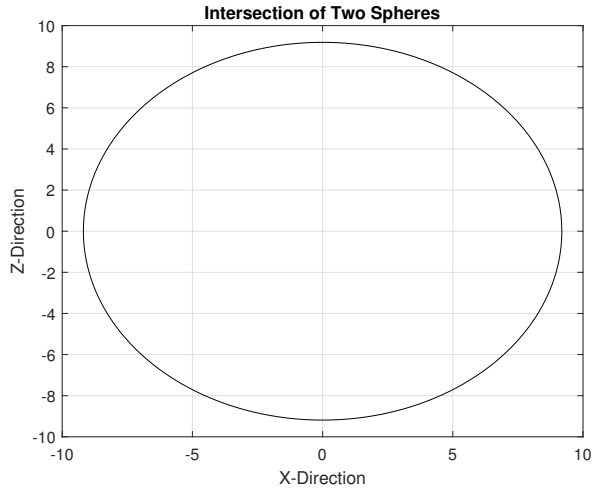


Figure 9.14: Circular Result of Sphere Intersection.

Z-direction being perpendicular from the ground. Any future stretch goal configurations will require the additional conversions. To find the intersection of the two spheres, EqnEqn 9.2 was subtracted from EqnEqn 9.2. Solving the resulting equation, EqnEqn 13.10, for the z variable. This results in the equation of a plane parallel to the XY-plane at the 'Z' height. The intersection of these two spheres and the plane that they intersect at is shown in 9.13. The height will be the calculated altitude for the application. By substituting this height into EqnEqn 9.1 and collecting the constants to define a new radius variable, the intersection of the sphere is defined by the equation of the circle in EqnEqn 9.7. The resulting circle can be seen in 9.14

The design generated these spheres by broadcasting radio waves and calculating the radii by utilizing time of arrival data due to the speed of the radio waves being constant, ie. the speed of light. To overcome difficulties with wireless communication implementations, UWB signals were used over the standard NB signals. A comparison of NB and UWB pulses can be seen in 9.15 [6]. The main difficulty with indoor application of wireless communication implementations is the distortion caused by random noise and multi-path interference. UWB signals allow for much narrower pulses than that of narrow band signals, this is the key feature when dealing with distortion. The effect of the distortion is quantized by the signal to noise ratio. This ratio can be seen in the Shannon-Hartley theorem, which shows that the rate at which information can be accurately transmitted with the presence random Gaussian distributed noise with a given bandwidth and signal-to-noise ratio, $R \leq \log_2(1+\frac{S}{N})$, where R is the bit rate, B is the bandwidth, $\frac{S}{N}$ is the average signal power received divided by the average power of the noise received known as the signal to noise ratio [7]. Multi-path interference is the interference introduced by the signal reflecting off of the environment and not direct line of sight propagation. Multi-path signals received have experienced a phase delay and an amplitude reduction that leads to destructive interference [7]. UWB technology also allows compensation for multi-path distortion by means of the Shannon-Hartley theorem. This is achieved by treating the multi-path signals as noise.

To overcome this a very large bandwidth is required which makes UWB technology ideal for this application. Due to noise introducing additive interference, high frequency signals are less susceptible due the rapid change the signal to noise ratio is larger. This can be seen in 9.16, the additivity of noise effects the UWB signal less because the narrower pulse peaks much faster than the NB pulse [6]. Narrower pulses also have a lower chance for experiencing the multi-path interference due to the lower probability of a multi-path signal to be received at the same time that a line of sight signal to be received. This is directly seen in 9.17 [6].

The benefit of implementing UWB radio modules was based on the utility of occupying a

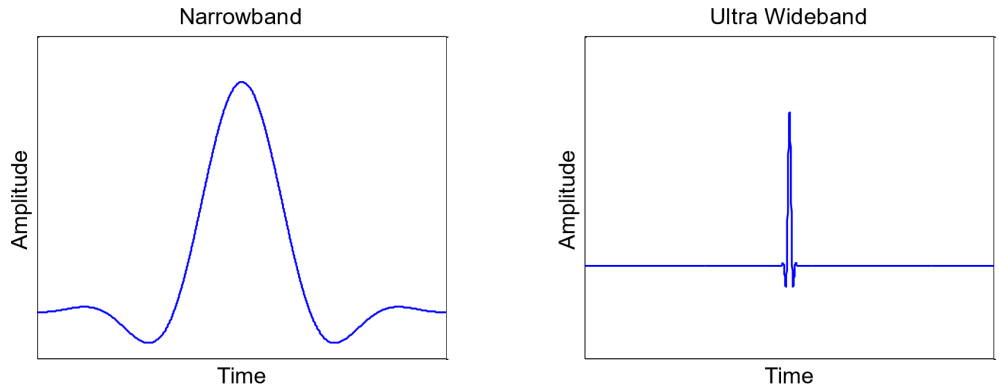


Figure 9.15: NB Pulse Versus UWB Pulse.

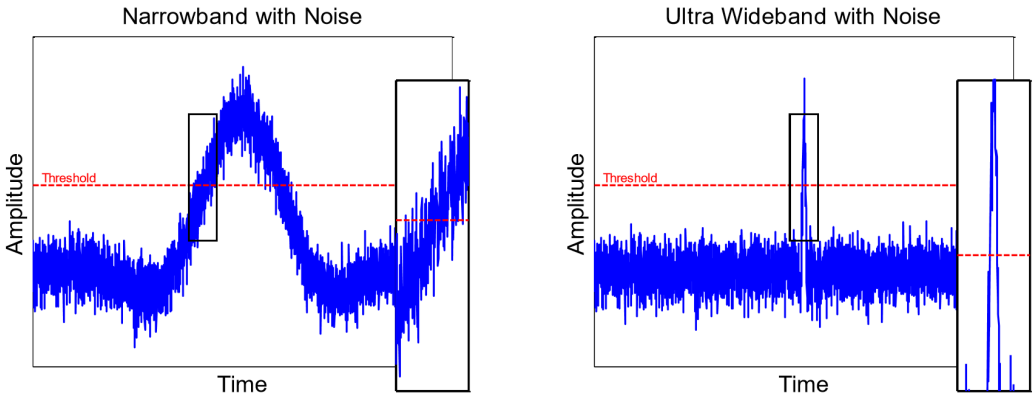


Figure 9.16: NB and UWB Pulses With Noise Interference.

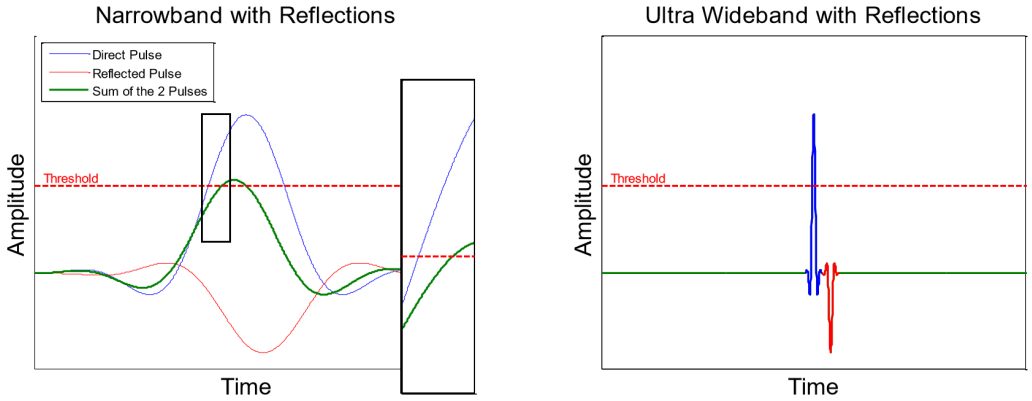


Figure 9.17: NB and UWB Pulses With Multi-Path Interference.

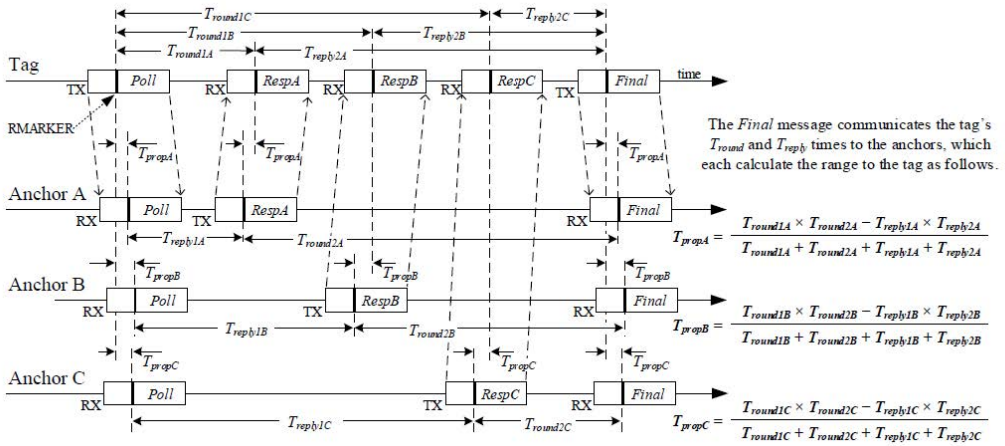


Figure 10.18: Depiction of UWB Ranging Algorithm.

large bandwidth. By taking advantage of the exchange of operating over a large bandwidth, UWB radio allowed for lower signal power, lower power consumption, and lower interference. UWB is an impulse radio system. This use of impulses requires a large band of frequencies. Generally the bandwidth of UWB envelopes a range that consists of 20% swing in both directions of the center frequency [7].

10. EXTRACTION OF UWB DATA (RAHUL)

In order to accurately develop a system for two UAS to match the position of one another, the position of each UAS must first be determined. In the absence of GPS, ultra-wideband radio modules are used to obtain the one-dimensional (1-D) position of each UAS in a local reference frame. These modules provide a sub-10cm accuracy, which is necessary when measuring the subtle movements of the UAS in a smaller reference frame. Before calculating the position of each UAS within the localized frame, it is necessary to obtain the direct distance between the UAS and each of the external UWB radios. Each radio contains serial communication via an onboard SPI bus. Data received by the radio via the IEEE 802.15.4 protocol can be transmitted via SPI to an on-board microcontroller, who then uses time of flight calculations to determine a range between two UWB radios (see Figure 10.18).

10.1. Ultra-Wideband Radio Modules

In order to eliminate the need to develop ranging calculations, the Decawave TREK1000 UWB ranging module was purchased with an onboard microcontroller and ranging algorithm.

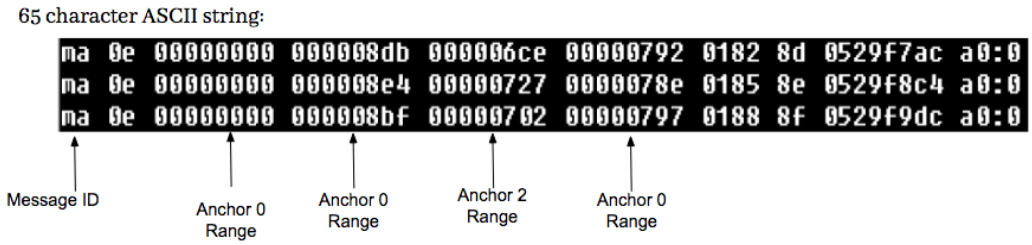


Figure 10.19: Output UWB stream via USB port.

This module allows for ranging between multiple UWB radios, and communication via onboard USB. Each TREK1000 unit includes an STM32F105, an ARM Cortex M3-based microcontroller that communicates with the UWB radio over SPI, calculates its distance from a set of radios, and outputs that data via an onboard USB serial port. Each radio module is configured to act as either an anchor or a tag. Anchors are capable of ranging with only one other device, whereas tags can range with multiple devices. It is assumed that the ”tag” configuration will be used on each UAS, while an ”anchor” configuration will be used on each satellite module. This configuration can be set via a series of hardware switches located on the back of the device. When configured as a tag, the UWB module is designed to cycle through and send a poll message to each of the available anchors. Upon receiving a response from each anchor, the module processes the time-of-flight to determine an approximate range to the module. The range to each of the available anchors is formatted as an ASCII text string, and streamed via the USB serial port located on the module. Figure 10.19 shows an example line of text as streamed through the device USB port. The UWB tag sends out a series of three different ASCII text streams, each denoted by a different message ID. ”ma” ranges provide the distance from anchor to anchor (the ”satellites” in our given experimental design). ”mr” ranges provide a raw anchor-to-tag range. These ranges are uncorrected, do not rely on past range data, and often times can lead to many erroneous data points. The third data stream consists of ”mc” ranges, or corrected ranges. These values are adjusted based on past data points to provide a more accurate range measurement between a tag and a given anchor. For the purposes of trilateration and UAS ranging, only the ”mc” data values are utilized.

10.2. Python Text Extraction

Using Python, a script was developed to run on the onboard Raspberry Pi CPU to receive all the data transmitted over the USB COM port of the UWB module. This script implements

Python's built-in PySerial package, which allows for connection between a computer and a serial device (e.g. a USB peripheral) at a user-specified baud rate. This script imports every line as output by the UWB module, and parses the line for the appropriate ranging information. The Message ID, always located in the first two characters of each ranging message, is read to determine the type of message. Because we are primarily concerned with tag-to-anchor distances, all "ma" messages are discarded (it is important to note that while the system checks for "ma" messages, they do not appear in the serial stream as the data is being output via a tag module, as opposed to an anchor). In addition, the system also checks the text stream for all "mr" values, and removes these values as well. Upon filtering, the only messages that remain contain corrected anchor-to-tag ranges. Because the structure of each message is identical, it is possible to simply extract specific segments of the character array to retrieve the ranges of specific tags. Namely, the distance from the tag to a predetermined Anchor 0 is always located in position [6:13], the distance to Anchor 1 is always located at position [15:22], etc. These segments were then extracted and saved to be for use in trilateration calculations.

10.3. Sampling Frequency & Erroneous Data

Using the Python text extraction method, which utilizes the TREK1000's built-in text output and ranging calculations, it is observed that data is saved to a text file at a rate of $10Hz$. This sampling frequency compares to that of GPS, which also samples at a rate of $10Hz$. However, due to the speed of the USB serial port on board the Raspberry Pi computer relative to the output stream of data from the TREK1000, it was observed that certain data packets were sometimes missed during the output stream. Due to the characteristics of the Pyserial "readline" function, partial data lines would arrive via the USB port, and were unable to be properly processed due to the offset in array indices of the characters in the stream. As a result, a series of software checks were implemented to run on every line that was sent over the USB serial port. The first of these checked the value of the initial character in the stream. If the value of the character was not an "m," it was clear that the received packet did not contain a full text stream, as every packet begins with an "m" value. In addition, to further ensure that packets did not arrive with missing or extra data, the packet length was also verified, to ensure a length of 65 bytes (or 65 ASCII characters). All erroneous packets were then discarded and not used in the accumulation of the digital filter and subsequent position calculation.

It was observed that a majority of erroneous data packets arrived during the first second of execution, after which point very little data was erroneous. It was then observed that after a

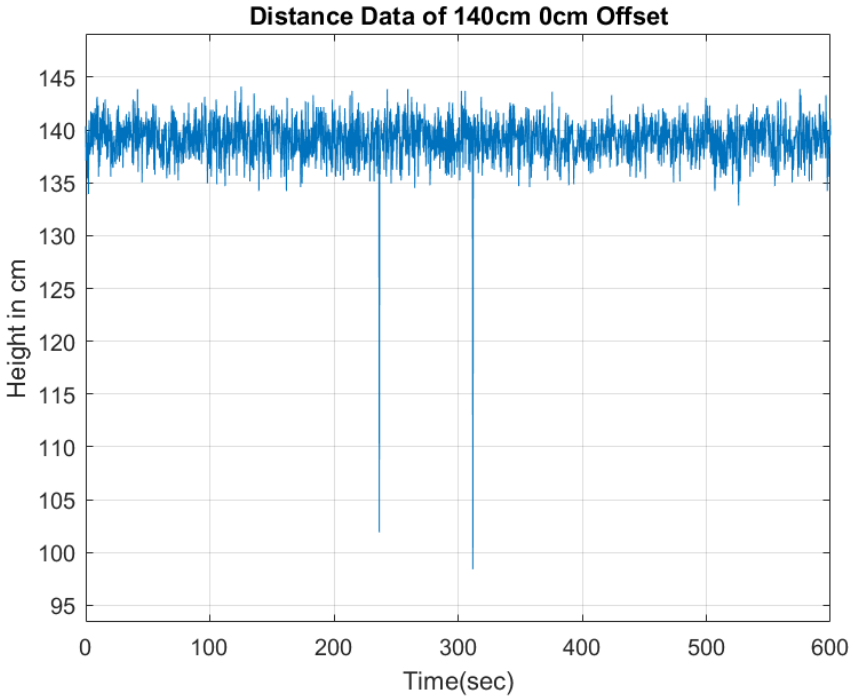


Figure 11.20: Distance Data of 140cm, 0cm Offset.

reasonable execution time, the packet drop rate only increased to 1%, which did not impose a significant effect on the output of the position data after processing via the digital filter.

11. UWB DATA PROCESSING (REECE)

An initial static test was performed with the UWB modules to see the response and if there were any errors in the ranging data. The testing was performed inside a cement building with the modules secured to the floor. The experimental setup consisted of two anchors place on the ground a distance of 247.5 cm away from each other, a tag placed between the anchors 140 cm away from anchor zero, and zero cm perpendicularly from the line of sight path of the two anchors. Data was collected for 10 minutes, this process was repeated for a perpendicular distance of 45 cm and 80 cm. The results were brought into MATLAB and adjusted accordingly for the time of 10 minutes. The plots can be seen in Figure 11.20, Figure 11.21, and Figure 11.22 respectively. As can be seen from the three figures, the signal bounces roughly 5 cm on either side of the 140 cm line, and that the further away from the line of sight path between the two anchors there was an increase in erroneous data points, two from Figure 11.20, three from Figure 11.21, and four from Figure 11.22. The standard error can be attributed to noise and the erroneous data point most likely point to multi-path interference, which both would have a

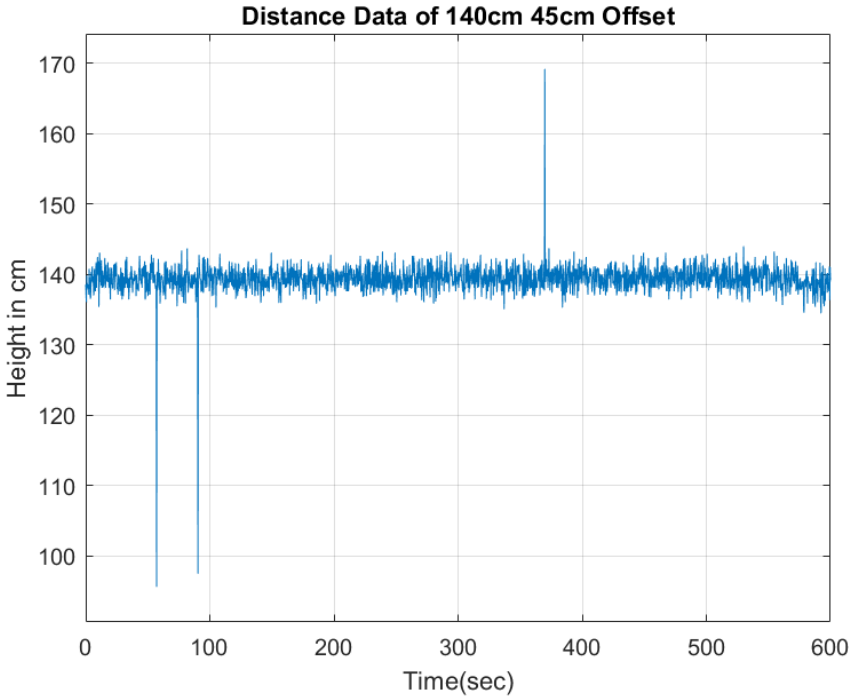


Figure 11.21: Distance Data of 140cm, 45cm Offset.

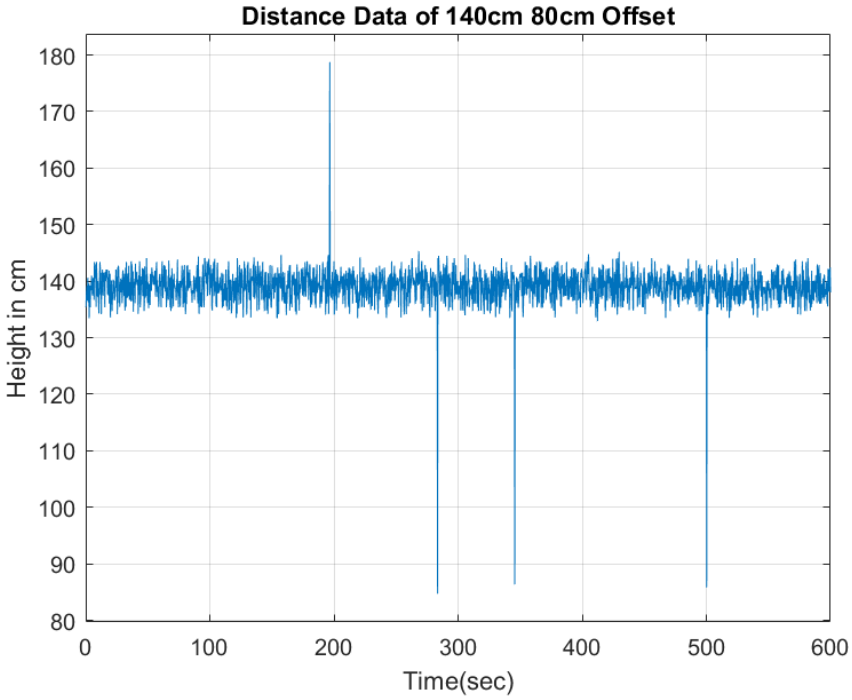


Figure 11.22: Distance Data of 140cm, 80cm Offset.

greater effect on the data acquisition due to the modules being indoor and so close to the floor. While there are other possible causes to the inaccuracy within the data set, the decision was made to implement a solution to treat the symptom and not the cause by implementing a digital filter.

11.1. Background (Reece)

The main effect of the digital signal processing is essentially that of a moving average. This is beneficial because a moving average can be modeled as a low pass filter (LPF). The two options for implementing digital filters are the finite impulse response (FIR) and the infinite impulse response (IIR) filters. FIR filters are mostly implemented through convolution. Implementation through convolution can be a time and resource sink from a computing perspective hence FIR filters are better for implementations where finite data sets are being processed. While IIR filters do have the possibility of going unstable due the fact that they typically have a z pole lying on the unit circle on the complex plane and are more susceptible to quantization error, the choice was to implement an IIR LPF due to the filter being able to be implemented as a difference equation and that IIR filters are derived from analog filters, which is better suited for this systems implementation due to the system operating as having an infinite data set. The three filter options are the Elliptical, Chebyshev, and the Butterworth filters which are analog designs but are made to be digital through a bilinear transform. The main appeal of the Elliptical design is that out of the three is does have the sharpest frequency cut off region but it is also the most mathematically intensive. Chebyshev filters allow for an equiripple in the passband, equiripple meaning that the gain throughout the pass band is effectively equal for all frequencies within the band. While this does make, the Chebyshev design a preferred it is also mathematically intensive. The third design is the Butterworth filter. The Butterworth design while still being highly effective is the simplest of the three, the most easily implementable, and the least computationally intensive leading to less delay due to processing. This is why the Butterworth design was chosen. The bilinear transform is an operation that utilizes the sampling rate of a digital signal to convert an analog filter to a digital filter. This is effectively accomplished by equating the analog frequency range of $-\infty < \omega < \infty$ to that of the digital frequency range $-\pi < \omega < \pi$. [8]

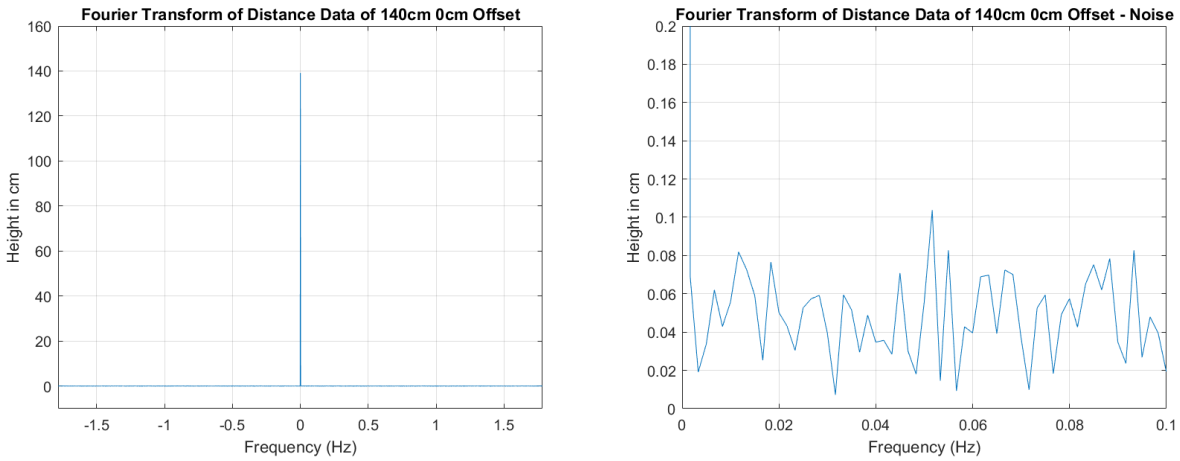


Figure 11.23: Fourier Tansform of Distance Data of 140cm with a 0cm Offset.

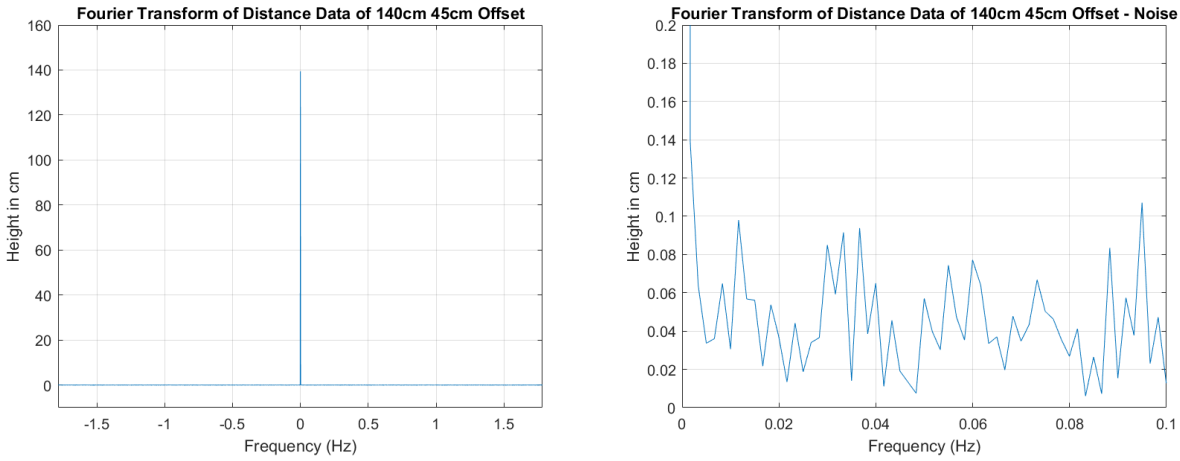


Figure 11.24: Fourier Tansform of Distance Data of 140cm with a 45cm Offset.

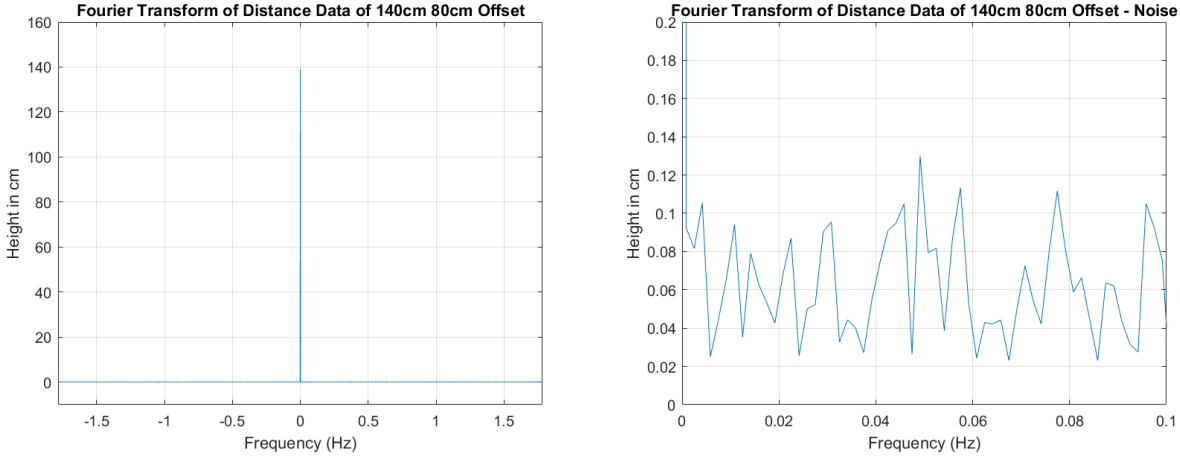


Figure 11.25: Fourier Transform of Distance Data of 140cm with a 80cm Offset.

11.2. Design (Reece)

Before starting the initial design of the filter, the three data sets in Figures 11.20, 11.21, and 11.22 were Fourier Transformed to be able to view the frequency content of the signals. The results are shown in Figures 11.23, 11.24, and 11.25 respectively. The resulting data shown in the figures agrees with what was expected. High in magnitude DC equivalent content at the lowest frequencies and essentially the high frequency content magnitude carries no weight. This leads to designing the LPF to have a very small bandwidth. The next consideration to take into account is what limitations are on the cutoff frequency. Due to the Nyquist criterion, $\frac{f_s}{2} > f_c$, stating that the filter cutoff frequency must remain less than half of the sampling frequency. Considering that the sampling frequency is roughly 4 Hz, the result is that the filter cut off frequency must remain below 2 Hz. The order of the filter must also be looked at. Due to higher order filters requiring longer processing time to output data, and the control system constraint of a settling time of 0.5s.

With taking these design considerations into account the actual filter design can be accomplished within MATLAB. By utilizing the MATLAB function 'butter'. For the purpose of implementing a LPF, the function takes in three inputs, the order of the desired filter, the cut off frequency normalized within the values of 0.0 and 1.0 corresponding to half of the sampling rate, and a string of low to define the filter as a LPF. The function then returns the two arrays. The first being the filter transfer function numerator coefficients based on z with descending powers. Z being the variable of the z-transform, but for this application will be treated as $\exp(j\omega)$ to represent the discrete time Fourier Transform. The second being the same as the first with the exception that it is the denominator coefficients. By taking these two resulting arrays and converting them into polynomials of z and the definition of the transfer function being, $H(z) = \frac{Y(z)}{X(Z)}$, the difference equation will be solve for by cross multiplying and taking the inverse Fourier Transform.

The initial results of the filter design give the impression of being incorrect due to filtered data requires a time period to reach the 140 cm steady state value. The result of filtering the original three data sets can be seen in Figures 11.26, 11.27, and 11.28 respectively. The current interpretation is that an error has been made in transforming the data sets into the frequency domain or on how the data set was converted to a time signal.

After the initial filter design, dynamic test was preformed to verify the filter and compare

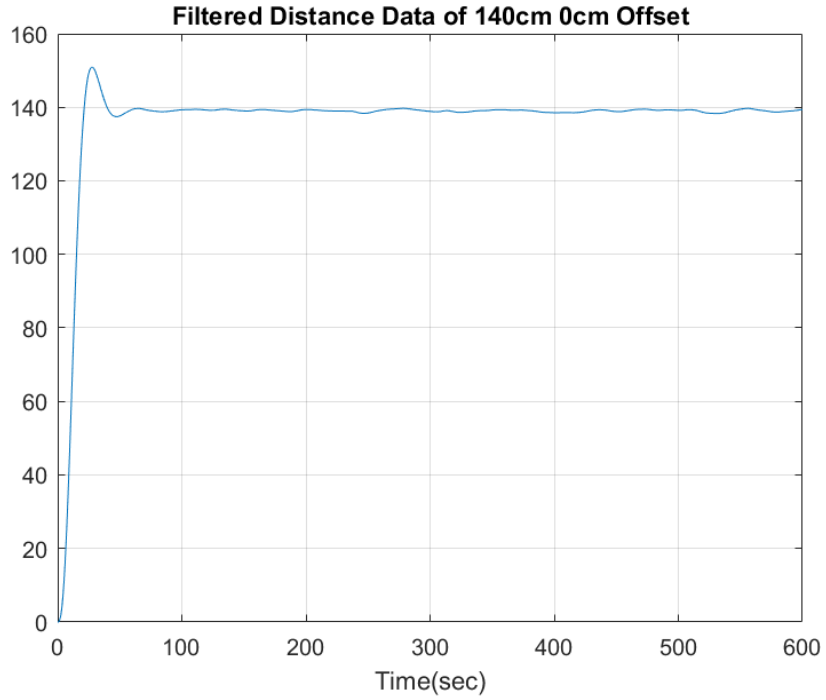


Figure 11.26: Filtered Distance Data of 140cm, 0cm Offset.

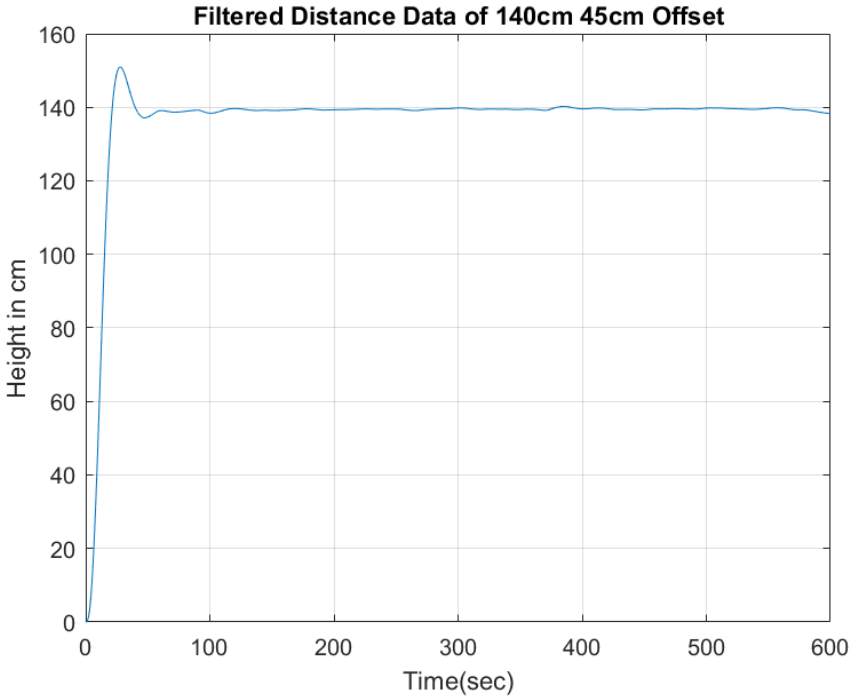


Figure 11.27: Filtered Distance Data of 140cm, 45cm Offset.

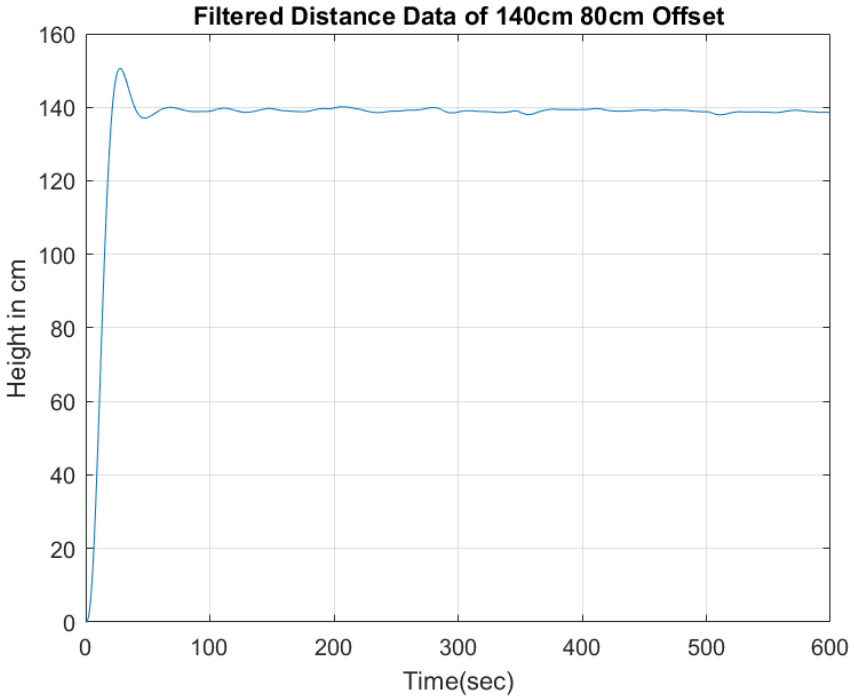


Figure 11.28: Filtered Distance Data of 140cm, 80cm Offset.

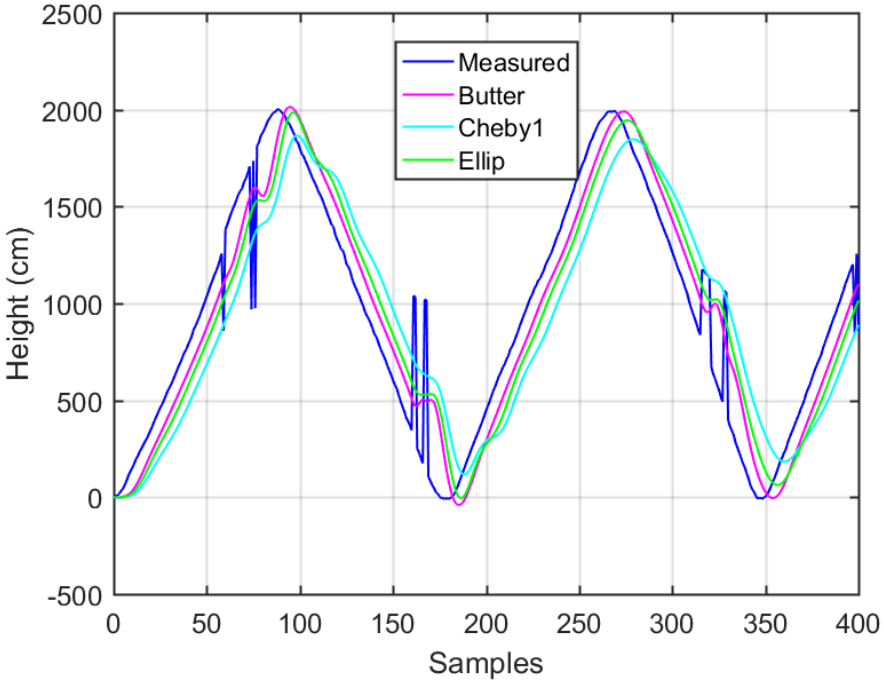


Figure 11.29: Dynamic Filter Test.

to compare the Butterworth, Chebyshev, and Elliptical filter types. The test was preformed by setting up the UWB anchor modules 20 meters apart and walking a tag UWB module back and forth. The results can be seen in 11.29. The results show that the Butterworth design is the most optimal. This is due to the Butterworth having the least phase delay.

12. EXPERIMENTAL SETUP FOR DISTANCE DATA (REECE)

The initial design of the system was to design the system such that the leader UAS would be flown to a certain altitude in the z-direction and set to hover, while the follower UAS would autonomously fly to match the leaders altitude height. Due to safety concerns with the disabling of any GPS data, the decision was made to switch for altitude match to x-direction matching. This allows for the UAS to be able to retain safety features as return to land and land. With this being taken into account the experimental setup and testing of the system had to be reconfigured. Now two anchor UWB modules will be placed at the top of two 6 foot ladders separated by 20 meters. Initial testing of the collecting of distance data from the UWB modules will be static by manual setting up a tag between the two anchors at various distances and verifying the data with the GPS locations of the two anchors and the tag with a mobile cell phone. The next test will consist of the same anchor set up but with the adjustment of the tag being mounted on a UAS and its GPS being verified by the GPS module on the hexacopter. This second test will also be used to verify the filtering of the UWB distance data.

13. POWER BOARD (REECE)

13.1. Background (Reece)

Due to the type of system being design the decision was made to separate the power distribution systems of the UAS itself and the peripheral measurement devices in a said science package. This is a benefits in protecting the science package from any noise produced from the operations of the UAS.

To protect the science package of peripheral modules, Ultra Wideband (UWB) radio modules and the Xbee modules, and to isolate the main circuit and the peripheral circuit from introducing interference between the two circuits a separate power board was needed within the design. The restrictions on the power board were to meet the power requirements of the science package, have a high efficiency rating, and the smallest footprint possible. These restrictions are to create a subsystem to provide reliable and constant power throughout a UAS flight to the science

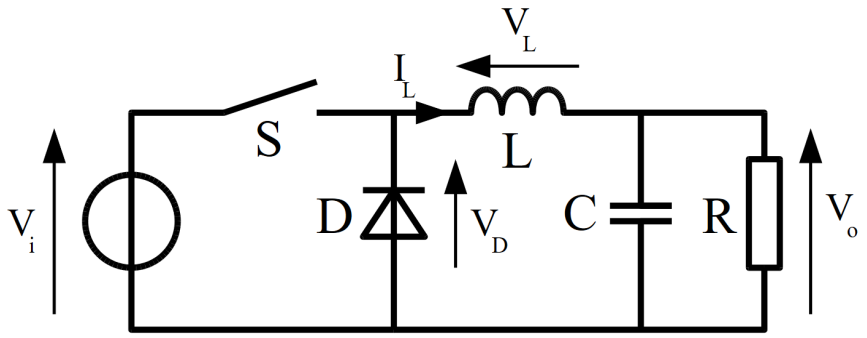


Figure 13.30: Basic Buck Converter.

package, and to have the most minimal effect on the mechanical performance of the hexacopter. Mechanical performances such as maintaining the center of gravity as close to possible to the physical center of the UAS.

To power the two circuits separately, the main unmanned aerial system will be powered by a three-cell lithium polymer (LiPo) battery while the science package power board was designed such that it will allow for an input voltage range of 9 to 15 V. The power requirements for the science package peripherals consist of suppling a voltage of 3.3 V and 250 mA to the UWB radio modules and 40 mA to the Xbee modules.

Under the design constrictions to maintain a small footprint and higher efficiency, the DC to DC power conversion would be performed by a buck converter circuit compared to the other options available. Voltage dividers, while simple to implement are highly inefficient and are not meant to be implemented to provide voltage to a sub circuit. They are more useful for applications where a reference voltage is needed such as when voltage measurements are needed and are not designed to allow for much current draw. Linear regulators are electronic circuit systems that are effective at maintaining a constant output voltage. The benefit of these regulators is that they have a linear behavior and are easily implemented. The main drawback of them is that to maintain the constant difference in output to input voltages, the difference is dissipated as heat which greatly limits the efficiency of the circuit since this energy released as heat is effectively just wasted and cannot be utilized. The third option was to use a buck converter circuit, as seen in Figure 13.30. [9]

$$Vout = D * Vin \quad \text{(Eqn 13.8)}$$

$$\Delta i = \frac{Vin - Vout}{L} x \frac{D}{fs} \quad \text{(Eqn 13.9)}$$

A buck converter consists of a switch, a diode, inductor, and a capacitor. Buck converters allow for higher power efficacy due to limiting the amount of time that the input voltage is effectively connected to the circuit. The switch is implemented with a transistor. The output voltage becomes a function of the input voltage and the duty cycle at which the switching transistor is being switched on and off, as seen in EqnEqn 13.8. The inductor acts as a current choke to stabilize the output current, while the capacitor stabilizes the output voltage. Both acting as low pass filters, the inductor as a current filter, and the capacitor as a voltage filter. With certain electronic implementations buck converter voltage regulators allow for different input voltages and rely on a reference voltage on the output side of the circuit to maintain the correct output voltage. The reference due to not needing a current input but only a voltage can be implemented with the aforementioned voltage divider circuit. Due to the specifications of the buck converter being the most optimal choice for the DC to DC voltage conversion, the decision was made to implement a Buck converter which allow for higher power efficiency compared other DC to DC converters such as voltage divider circuits and linear regulators. [9]

To meet the design criterion of a small footprint, the power board system was to be implemented in a printable circuit board. This is beneficial due to the majority of the circuit being implemented in electronics the footprint of the buck converter regulator is small and can be reduced further by implementing the other components in surface mount variations. This design fits the project system due to keeping a reduced payload and footprint that will allow for maintaining the center of gravity of the hexacopter.

13.2. Design (Reece)

To begin the design of the buck converter power board, a rough design was implemented with a N-channel MOSFET utilizing EqnEqn 13.8 to find a duty cycle of 0.29, and EqnEqn 13.9 to get initial values for the switching frequency and the inductance. Using these values and a

capacitor to limit the ripple output voltage were used to simulate the design in PSIM, a power simulation software. The results verified the design while being implemented while maintaining the calculated duty cycle and a frequency of 500 kHz. Trying to narrow down the list of possible buck converter chips was reduced based on this behavior limited the options. Due to the industry standard of low voltage buck converters being built with an internal oscillator to provide the switching frequency, the original design had to be redesigned. To narrow down the possible buck converters and to meet with the design criterion, the online software tool TI Webench by Texas Instruments was used. The software allows for input, output, and load criterion/ parameters to be set and provides possible designs and available hardware options. This online tool also allowed for designing a circuit with a higher efficiency due to the hardware options being listed by efficiency ratings.

Due to having the highest efficiency rating of 0.95, the decision to use the TPS5430DDC 4.5 V to 28 V input synchronous step-down converter was made. The circuit designed can be seen in Figure 13.31. Through hole components for the circuit were obtained and the buck converter circuit was prototyped on a bread board, due to the buck converter chips being surface mount it was prototyped by soldering the chip to a breakout board. To simulate the UWB modules and the Xbee modules for a bench test, the 3.3 V output voltage was divided by the total operating current of the two modules, 250 mA and 30 mA, to find an equivalent test resistance. This resulted in a resistance of 11.8 Ω. A nominal resistance value of 12 Ω was used to represent the two loads. The test was also performed with the 12 Ω resistor. Using a DC power source as the input voltage to the circuit and stepping the voltage in increments of 0.5 V starting at 9 V until 14 V. The results of the bench test can be seen in Table 1. As can be seen from the results, the design did not perform as designed. To discover any possible cause for the unpredicted behavior, the circuit connections were tested and verified to be connected with the correct orientation, and to have strong connections. From adding resistance to the load it was discovered that the output voltage did not stabilize until 200 Ω.

$$R_crit = \frac{2Lf_s}{1-D} \qquad \text{(Eqn 13.10)}$$

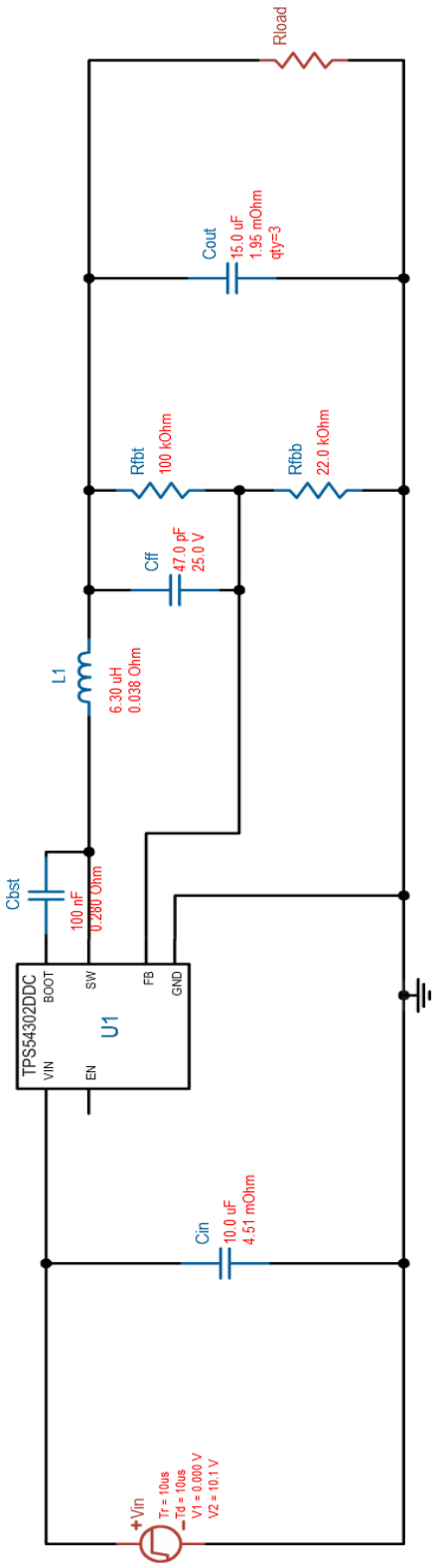


Figure 13.31: TPS5430DDC Buck Converter Circuit.

Table 1: TPS5430DDC Bench Test Results

Input Voltage (V)	Voltage Across Test Load Resistance of 12Ω (V)
9	4.5
9.5	6.4
10	8.3
10.5	8.8
11	8.8
11.5	8.9
12	8.9

After more research, it was discovered that there is a minimum critical resistance value that must be met for a buck converter to operate properly. The equation for this minimum load resistance can be seen in Eqn 2.Eqn 13.10. When the load resistance falls below this resistance the output voltage approaches the input voltage. Using equation Eqn 13.10, the critical resistance was found to be 8.8 Ω. While the test load resistance is above this minimum resistance from eqn Eqn 13.10 is for the basic buck converter design. Due to the TPS5430DDC being more advanced in electronics, this equation is not the only factor to take into account. By returning to the TI Webench software, the parameters were adjusted to take into account for this. But this resulted in an increase in a decrease in the size of the inductor, which in turn resulted in a peak output current that would pose a possible danger in causing damage to the UWB modules and the Xbee modules. This lead to a redesign of the circuit.

A common trend discovered was the more efficient converter chips require a greater load resistance to maintain the buck converter in the desired mode of operation. Research was done into more commonly used converter chips and the TI LM2576 Simple Switcher 3-A Step-Down Voltage Regulator was discovered due to the regulator allowing for higher output currents at the cost of a reduced efficiency. Using the TI Webench tool the buck converter circuit was designed using the LM2576 buck converter. This resulted in a simulated efficiency of 0.80. While this is largely reduced from the 0.95 efficiency of the first design, with the correct choice in supply voltage battery with the correct energy storage and discharge rating. The battery chosen was a LiFe 9.9V battery, with ratings 1500 mAh and a discharge rate of 1C. This will be able to provide 300 mA for roughly 5 hours. This is beneficial due to the hexacopter flight time being roughly 15 minutes, this allows for several flight tests without the need to recharge the battery.

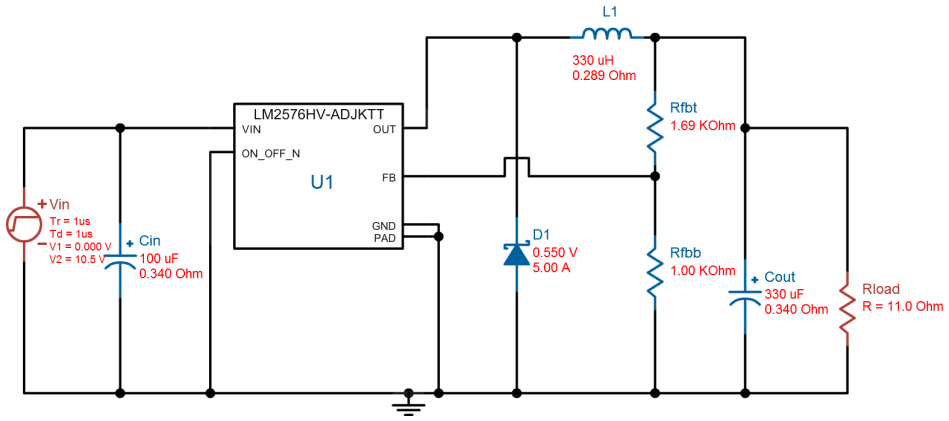


Figure 13.32: LM2576-HV-SX

Table 2: LM2576-HV-SX Bench Test Results

Input Voltage (V)	Voltage Across Test Load Resistance of 12Ω (V)
9	3.3
9.5	3.3
10	3.3
10.5	3.3
11	3.3
11.5	3.3
12	3.3

The second circuit was simulated through TI Webench and the output voltage and output current steady state values were calculated and found to be 3.3 V and 0.3 A respectively. While this result matches the design, the physical system needed to be tested to fully verify the design. The schematic of the design with the component values labeled can be seen in Figure 13.32.

To bench test the new power board design, a through hole versions of the LM2576-HV-SX and the components shown in figure 13.32 were obtained. The prototyped circuit was built on a bread board. The circuit was tested as the first circuit was tested, using the DC power source as the input voltage and the voltage was stepped in increments of 0.5 V starting at 9 V until 14 V, with the difference that it was only tested with the single 12 Ω resistor. The results can be seen in Table (Design 2).

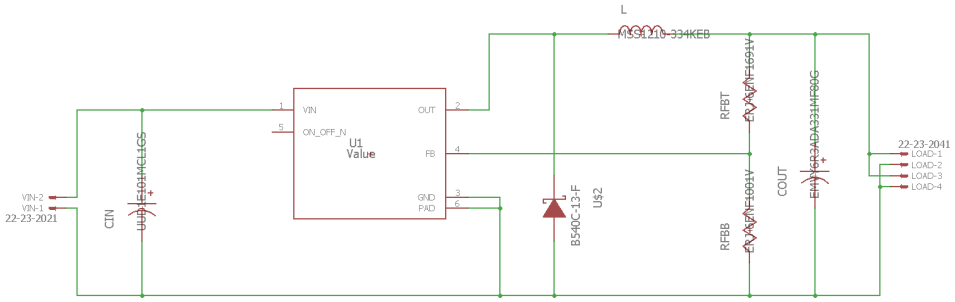


Figure 13.33: LM2576-HV-SX Buck Converter Circuit Schematic - Eagle CAD

The results from Table 2 verify the designed circuit with the LM2576-HV-SX buck converter. The transient response of the circuit was also measured on the oscilloscope. The max peak voltage was measured to be 3.948 V and therefore the max current was 329 mA, both which are within the ranges of the modules. The next step in the design was to design the circuit in a printed circuit board (PCB). This is required based from the design requirements to reduce the size of the power board. By achieving this the secondary power system will have a smaller effect on the mechanics of the system. To do this the power board circuit was implemented within the design program Eagle CAD. Eagle CAD is a computer aided design program for the design of PCB systems. The program works a s a scriptable electronic design application. Eagle CAD has the features for circuit schematic capture, PCB layout, manual and auto-routing of the board connections. One of the most effective built in features of the Eagle CAD program is the interconnection between the schematic capture and the PCB layout features. When a circuit has been designed within the schematic capture, the design can be brought into the PCB layout editor with highlighted interconnections between components mirroring the connections shown on the circuit schematic. This permits an ease when placing circuit components on the PCB layout.

The circuit was designed within the schematic capture as shown in Figure 13.32, and can be seen in Figure 13.33. Eagle CAD does have built in libraries but these are limited and selective to certain brands of components. This requires that a project specific library be created for the model components of the power board to match the physical components. To bring the components of the circuit and to create the surface mount footprints, a library script file was exported from the TI Webench design. There was difficulty in running this library script due to the documentation being concealed within a readme file created from the export from TI

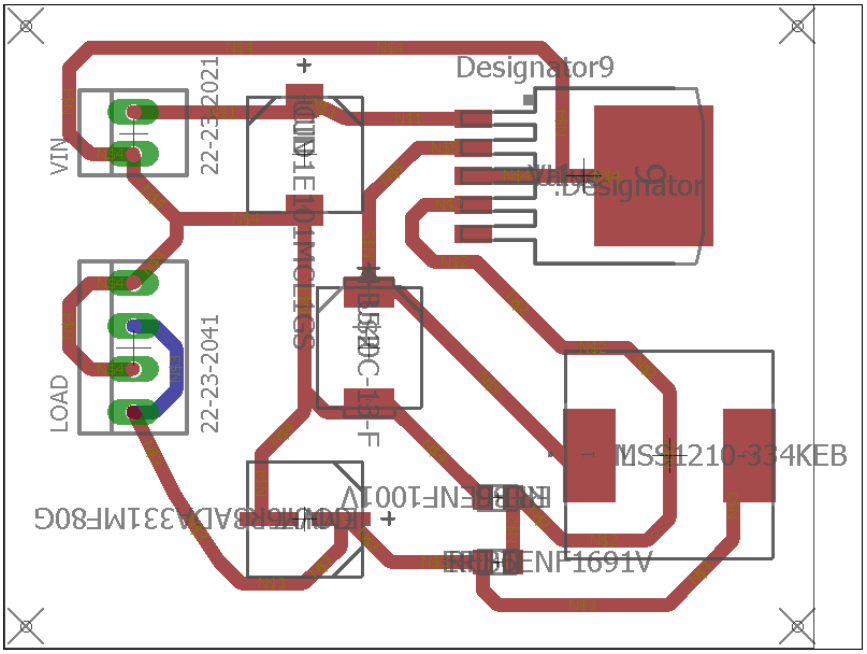


Figure 13.34: LM2576-HV-SX Buck Converter PCB Layout - Eagle CAD.

Webench. The library script was ran by starting at the Eagle CAD control panel, selecting from the menu File New Library, to open the library window. From the library window, selecting from File Execute Script, and then selecting the library script file from the power board project.

The main design considerations that were made with the PCB design is the placement of the circuit components, the trace spacing, and the angles at which the traces bend. The main consideration the applies to all three of these criterion is the spacing. The PCB layout must allow for enough space for various reasons. The placement of the circuit components must be placed on to the board with the consideration of the thermal effects and provide the required space for the traces. Due to the high switching frequency of the buck converter, the component will produce a considerable amount of heat. With this in mind, the buck converter was placed on the corner of the board with the grounded heat sink facing away from the center of the board. Using the highlighted interconnection feature, the other components were placed to give a wide enough breadth for the traces. For the connection of the input voltage and an output for load connection, standard through hole Molex connectors were placed on the board. The input a two pin and the output a four pin. The choice of the Molex connector was due to the ability to use both a Molex connectors and a direct wire connection. The benefit is that Molex connectors are a standard connector that will allow for quick switching of power sources and the ability to make a direct

wire connection if the Molex connectors are not obtained within the time line of testing. With the placement of the components the PCB board was determined to be 2 x 1.5 square inches. To connect all of the components on the PCB layout the auto-routing tool was used, with the setting of a trace angles of 45° and a trace width of 0.032 mm. This is due to possibility of trace angles close to 90° causing electromagnetic induction leading to electromagnetic interference. The interference has the potential to be detrimental the system as a whole, due to the system requiring several communication lines, UWB, Xbee, and transmitter/ receiver for manual control. The increase of trace width is due to the limitations of the companies that preform the production of the PCBs. With the results of the auto-routing causing an implementation of four vias to jump across the bottom layer of the PCB, the connections were reduced leading to no need of via connections and only one connection across the bottom layer of the board. To allow for the power board to be easily mounted on the hexacopter 5 mm holes were placed at each of the corners of the board. The resulting PCB layout can be seen in Figure 13.34.For the Eagle CAD program the power board '.brd' file was exported and used to order the power board PCB through OSH Park.

14. SYSTEM MODELING (RAHUL)

14.1. Theoretical Dynamic Model

The dynamics of a multirotor UAS system have been heavily modeled by previous work in this field [10] [11]. One such model analyzes the idealized dynamics of a quadrotor UAS, and its position in both a local and global reference frame [10]. A diagram of such a system can be seen in Figure 14.35 [1].

Each rotor produces a vertical force F_i defined by:

$$F_i = k_F \omega_i^2 \qquad \text{(Eqn 14.11)}$$

where ω is the speed of an individual motor.

The global reference frame *W* is defined by the axes x_W, y_W, z_W, where z_W points directly upwards. The body frame *B* is defined by the axes x_B, y_B, z_B, where z_B, where z_B points perpendicular to the plane of the rotors, and x_B points in the "forward" direction of motion. The

[1]Note: the presented mathematical model describes that of a quadrotor UAS, while the system used in this work is a six-rotor UAS. However, (Eqn 14.12) still holds, as the acceleration of the UAS center of mass is simply a function of the generic sum of rotor velocities/forces, irrespective of number of rotors

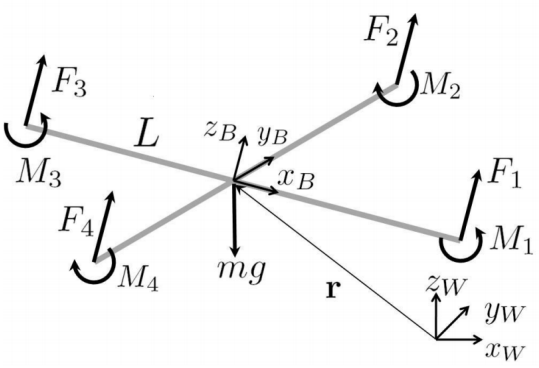


Figure 14.35: Model & Forces of Quadrotor System)

following rotation matrix *R* can be used to transform the motion and coordinates of the UAS in its local body frame to the global frame:

$$R = \begin{bmatrix} \cos\psi\cos\theta - \sin\phi\sin\psi\sin\theta & -\cos\phi\sin\psi & \cos\psi\cos\theta + \cos\theta\sin\phi\sin\psi \\ \cos\theta\sin\psi + \cos\psi\sin\phi\sin\theta & \cos\phi\cos\psi & \sin\psi\sin\theta - \cos\psi\cos\theta\sin\phi \\ -\cos\phi\sin\theta & \sin\phi & \cos\phi\cos\theta \end{bmatrix}$$

where θ, ϕ, ψ denote the pitch, roll, and yaw of the UAS relative to the global frame.

Defining the three-dimensional position vector of the center of mass by **r**, the net acceleration of the center of mass (or $\ddot{\mathbf{r}}$) can be defined by Newton's 2nd Law of Motion. the sum of these motor forces can thus define the linear acceleration of the center of mass by the following:

$$\mathbf{m}\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ \Sigma F_i \end{bmatrix} \qquad \text{(Eqn 14.12)}$$

The pitch, roll, and yaw can each be defined as double integrals of the angular velocity, which in turn is a function of the moments of the sum of motors, each given by

$$M_i = k_M \omega_i^2. \qquad \text{(Eqn 14.13)}$$

Thus, it can be determined that both the attitude and linear acceleration are functions of the angular velocity of the motors.

14.2. Physical Dynamic Test

If a multirotor UAS were to be developed from scratch for the development of this work, it would be necessary to test the physical dynamic characteristics of each individual motor, determine k_F and k_M to accurately determine the force and moment of each motor, and develop a controller for the UAS attitude (and transitively, its linear acceleration). However, this step is unnecessary, due to the existence of the Pixhawk 2 flight controller in the system. The Pixhawk 2 flight controller is a stably-designed autopilot system that can effectively maintain a given attitude and velocity through the use of its redundant inertial measurement unit (IMU) system and onboard controller. The system also uses its onbaord GPS to avoid drift, and with the combination of all three sensors, a PixHawk 2-connected UAS can effectively ”loiter” in any XYZ coordinate. With this system, a user is limited to only applying a velocity or a position to the controller, and the controller will then determine the necessary motor speeds to produce the desired acceleration that will thus produce the desired position or velocity. In order to develop an effective external position controller for the follower UAS, it is necessary to verify the dynamics of the system and verify that the physical system characteristics match the theoretical system characteristics as closely as possible. Namely, it is necessary to verify that the Pixhawk 2 does in fact reach given velocities when commanded to do so via an external controller. To do this, an experiment was designed that autonomously flew the UAS at a given constant velocity, and captured the position and velocity of the aircraft over time. Using the DroneKit Python library, a script was developed that autonomously starts, arms and takes off the UAS to a given height. From that point, the UAS is commanded to move in one direction at a given velocity for a set time duration, and then stop. While in motion, the GPS coordinates of the UAS are constantly logged to a text file at 0.2s intervals. This process is then repeated for multiple velocities and multiple time durations. This motion simulates a unit step function of the velocity of the UAS, which can be observed as a system step response for the UAS system. This test was carried out in an open field, such that the UAS had ample room to move without obstructions.

Figures 14.36 and 14.37 show the dynamic characteristics observed during the execution of the test. It is evident that, with the exception of slight nonlinearities throughout (likely due to wind and air resistance), the system reaches the desired velocity of 4m/s and maintains that velocity throughout the execution of the test.

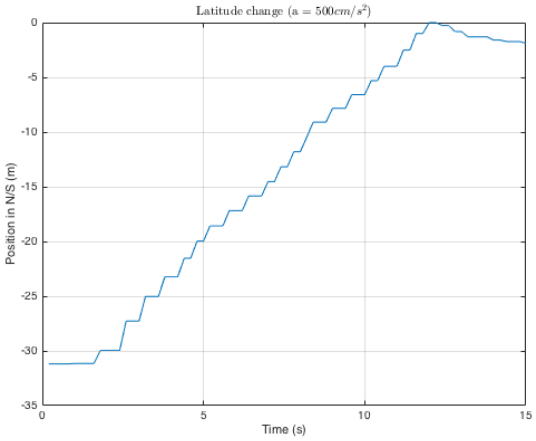


Figure 14.36: Position Change over 10s Interval ($\alpha = 500m/s^2$)

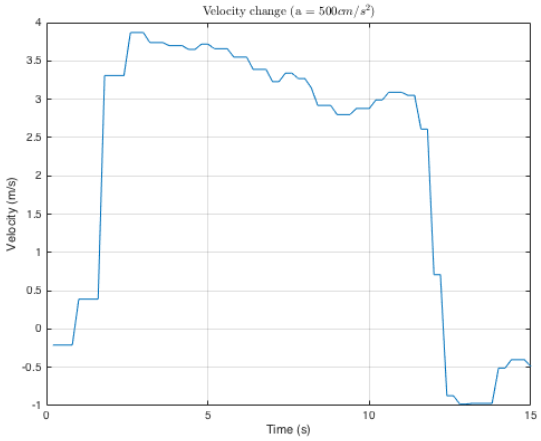


Figure 14.37: Velocity Change over 10s Interval ($\alpha = 500m/s^2$)

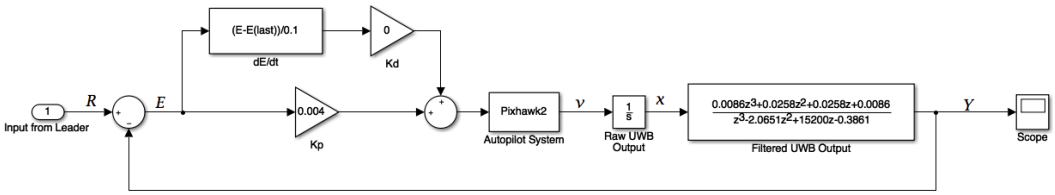


Figure 15.38: System Block Diagram

15. CONTROL METHODOLOGY (RAHUL)

For this system, a nested feedback loop-based control metholodogy was adopted. Figure 15.38 shows an overall block diagram of the system model. The system is split into an inner attitude control loop, responsible for controlling the speed of individual motors, as well as pitch, roll, and yaw, based on the desired position or velocity, and an outer trajectory control loop, responsible for controlling the trajectory and position of the UAS in the 1-D localized space. Because the Pixhawk 2 can effectively control the attitude of the UAS, it is prudent to continue to utilize the available controller, and to abstract each vehicle as a point mass. Through the use of the Pixhawk 2 autopilot, the attitude control of the UAS is complete, and the inner control loop can simply be abstracted as a generic ”Pixhawk2” system block on the diagram. It is now only necessary to design a trajectory controller for the UAS that will interface with the autopilot system.

The Pixhawk 2 (and included Ardupilot firmware) is capable of controlling both a user-defined position or velocity. When controlling position, the autopilot uses the on-board GPS

as its feedback signal to determine its system error. When controlling velocity, however, the autopilot solely uses its on-board IMU to determine system error. in order for the formation control to be truly GPS-void, the UAS needs to receive commands that are nonrelient on GPS. As a result, the input to the Pixhawk 2 (and therefore the output of the trajectory controller) is required to be a velocity.

Let $\mathbf{r}_T$ be the desired 1-D position of the follower (this position is, in fact, the position of the leader streamed to the follower), and let $\mathbf{r}$ be the current position of the follower (as measured by the filtered UWB module output). Thus, the error signal $\mathbf{e}$ is given by

$$\mathbf{e} = \mathbf{r}_T - \mathbf{r} \tag{Eqn 15.14}$$

The output of the vehicle (after passing through the attitude control loop) is a linear velocity. Once measured by the UWB module, however, the true system output is that of a position. It can thus be assumed that the UWB module (and associated infinite-impulse-response (IIR) filter) exist as a simple integrator and third-order transfer function located on the output of the Pixhawk. By the final value theorem,

$$\lim_{t\to\infty} E(t) = \lim_{x\to 0} sE(s)$$

it is known that the existence of an integrator in a closed-loop system eliminates the steady-state error of the system. Because an ”integrator” already exists in this system, only a proportional-derivative (PD) controller is necessary to control the position of the follower.

The desired velocity is calculated via the following:

$$\dot{\mathbf{r}} = k_p\mathbf{e}_p + k_d\mathbf{e}_v \tag{Eqn 15.15}$$

where $\mathbf{e}_v$ is the velocity error

$$\mathbf{e}_v = \dot{\mathbf{r}}_T - \dot{\mathbf{r}}$$

or, more precisely,

$$\mathbf{e}_v = \frac{d}{dt}(\mathbf{r}_T - \mathbf{r})$$

In a digital system, it is difficult to calculate a true derivative, so $\mathbf{e}_v$ was simply approximated as

$$\mathbf{e}_v = \frac{\mathbf{e} - \mathbf{e}_{prev}}{0.1}$$

where 0.1 is the sampling time in seconds between the current and previous value of $\mathbf{e}$.

Thus, the final system PD controller is:

$$\dot{\mathbf{r}} = k_p\mathbf{e} + 10k_d(\mathbf{e} - \mathbf{e}_{prev}) \tag{Eqn 15.16}$$

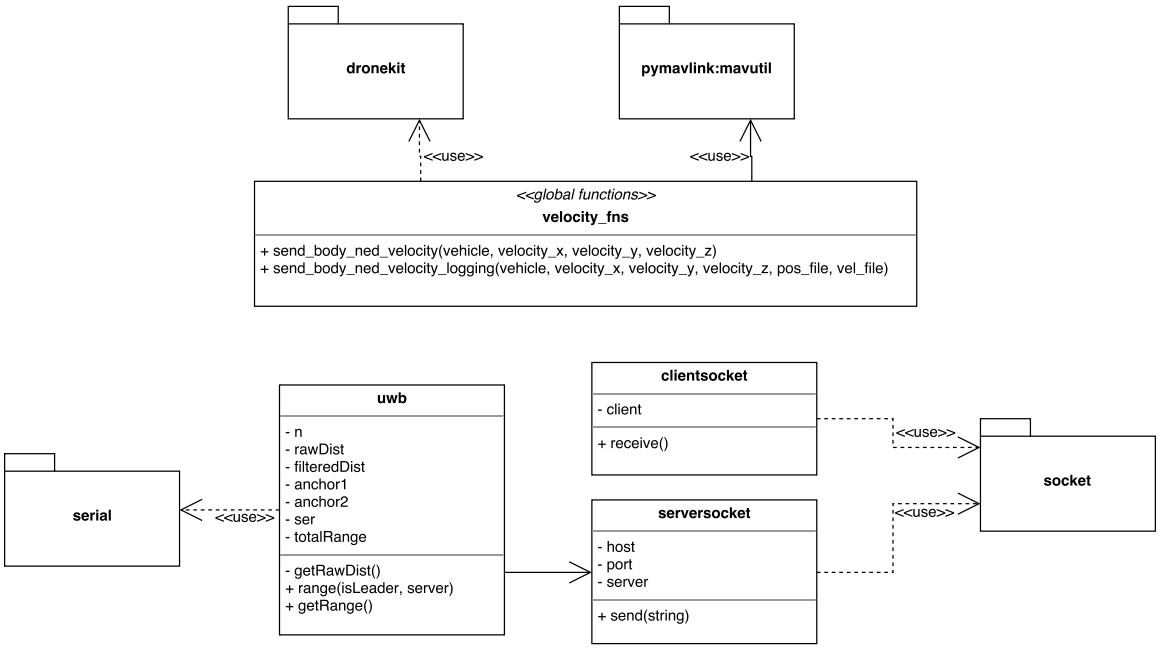


Figure 16.39: Class diagram of complete software system.

16. SOFTWARE IMPLEMENTATION (RAHUL)

Real-time control and monitoring of a single UAS is an incredibly difficult task, and the simultaneous control of a set of UAS presents an increased set of challenges. Namely, the integration of multiple real-time tasks requires a concurrent approach to the execution of tasks on-board each UAS. A multithreaded, object-oriented approach was adopted in Python for the implementation of the system on the external companion computer. Figure 16.39 depicts the class structure of the entire software system.

16.1. Velocity Functions & Custom MAVLink Messages

Because the UWB modules will be localizing each UAS in only one dimension, the UWB-based system is insufficient to be a viable replacement to the onboard GPS currently on the UAS. As a result, GPS is still active in the system, in order to avoid drift. The controller described above controls the position of the UAS in one dimension by sending a velocity to the Pixhawk 2. The companion computer utilizes the Dronekit library to transmit velocity commands to the UAS, and the controller described above adjusts the velocity of the UAS as it moves towards and away from its target (leader UAS).

MAVLink, a communication protocol designed for UAS and other autonomous systems, is the basis of all communication in the Ardupilot autopilot system. The DroneKit Python

library abstracts MAVLink commands for specifc tasks, and wraps them inside dedicated Python functions. The 3DR Dronekit library serves as a wrapper for MAVLink messages that are sent across the various systems. MAVLink is used as he primary communication layer between all components of the Ardupilot system. MAVLink directly links to the Ardupilot firmware and is the most direct communication between the software and each of the hardware peripherals. This makes the MAVLink protocol extremely powerful for sending programmatic commands to the Pixhawk via a companion computer.

Dronekit includes a MAVLink message factory, which allows for the creation of MAVLink packets through Python code. Because of this, it is possible to develop custom functions in Dronekit that can be used to interface with the UAS. One such function is that of velocity control. Dronekit does not inherently include a function to send a specific velocity to the UAS. However, MAVLink does offer this functionality, as it is necessary to control the velocity of a UAS when using a joystick or sending autonomous waypoints. As a result, a custom ”send_velocity” function was developed to add the custom functionality. This function receives four inputs: velocity in X, Y, and Z directions, as well as a pointer to the ”vehicle” object being controlled, and develops a text string that is passed to the MAVLink message factory. The MAVLink message is then sent across the serial connection between the companion computer and Pixhawk 2, and the UAS then moves at the desired velocity.

16.2. Multithreaded Implementation

The follower UAS must be capable of simultaneously measuring and calculating its own position, as well as interfacing with the Dronekit API to control its position with reference to the leader. Multithreading was used to execute both tasks simultaneously. The system utilized the Python ”threading” library in order to easily create, join, and switch between threads. Each function, that of the trilateration algorithm and the controller, was encapsulated inside an individual thread. The main thread creates both threads at the beginning of execution, and assigns each function to the thread. From that point, the OS and Linux kernel on the Raspberry Pi manage task scheduling such that both functions can run as close to simultaneously as possible. Because the controller must access data from the UWB radios as the position is being calculated, a separate ”getRange” function was developed that establishes a connection between the two threads.

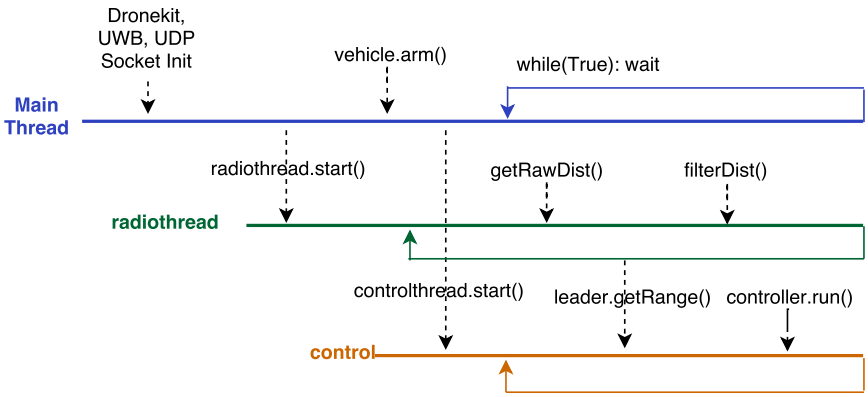


Figure 16.40: Multithreaded Sequence Diagram.

16.3. UWB Class

All functionality of the UWB radios, including the trilateration algorithm, exists in the ”UWB” class. This class contains three functions. The first of these calculates a raw position based on the outputs of the UWB anchors. This function utilizes the PySerial library to interface with the UWB module over USB UART. This function is private, and is only called by the ”range” function. The ”range” function takes in all raw values, and applies the developed digital filter to the data in order to avoid erroneous spikes or changes. This function executes forever, and is designed to run inside of a thread. The last function, ”getRange,” returns the most recently updated value of the array of ranges. This function is used by the controller to determine the most recent position of the UAS.

16.4. Client and Server Socket Classes

A *serversocket* class is developed to handle the instantiation and transmission of data from the leader UAS. This class extends the Python ”socket” library [12] in order to create customized functions specific to the transmission of position data to the follower UAS. The *serversocket* class includes two functions: a constructor and a function to send messages across the network. The *serversocket* constructor takes in a hostname (given as the follower UAS IP address) as well as a specific port on the IP, and assigns that hostname and port to the serversocket object’s private ”hostname” and ”port” members. In addition, it instantiates a socket object and assigns the object to the serversocket’s ”server” private member. This abstraction is developed as a matter of convenience, to eliminate the need to repeatedly direct access to the Python ”socket” library. The second function, the ”send” function, takes in data as a string. It then uses the socket library

”sendto” function to send the given string to the IP address and port specified by the socketserver object’s own member variables.

The *clientsocket* class functions similarly to the *serversocket* class. This class also extends the Python ”socket” library, and creates custom functions specific to the client (follower) device. The *clientsocket* constructor takes in both a host and port, however the host is left blank, as the client is designed to receive input from any server. However, the port that the client connects to matches that of the server, in order for both devices to send and receive data with one another. The constructor then creates a socket object called ”client,” and binds to the specified hostname and port. The *clientserver*’s ”receive” function simply encapsulates the socket library ”receive” command, and returns the data received.

16.5. Main ”Leader” Script

Because the leader UAS is manually controlled, and because all state information is retrieved from the external UWB module and sent over the local area network to the followers, it is not necessary for the software to interact with the Pixhawk autopilot system directly. As a result, the leader UAS is simply tasked with recording its own state in the 1-D localized space, and sending that information to its followers. Its two main tasks, interacting over the local wireless network, and calculating its UWB position data, are each encompassed in an individual class. The ”leader” script main thread creates an instance of both the ”uwb” and ”serversocket” classes, begins executing the UWB ”range” function in a separate thread, and then stays in a continuous wait state.

16.6. Main ”Follower” Script

The follower script contains all necessary functionality to interact and control the UAS, and integrates all position information from itself and leader to make decisions about its motion. The script begins by instantiating a Dronekit ”vehicle” object, a ”uwb” object, and a ”clientsocket” object. After connecting to the vehicle, it generates a new thread to begin recording position. This thread exists for the execution lifetime of the script. It After this time, the script sits in a wait state for 15 seconds, in order for the digital filter values to settle. It then ensures that the UAS is armed, and then generates a new thread to run an infinite ”control” function. This function implements the controller defined in Eqn 15.16 and executes in an infinite loop. It receives the leader’s position via the client ”receive” function, and accesses its own position via its uwb ”get” function.

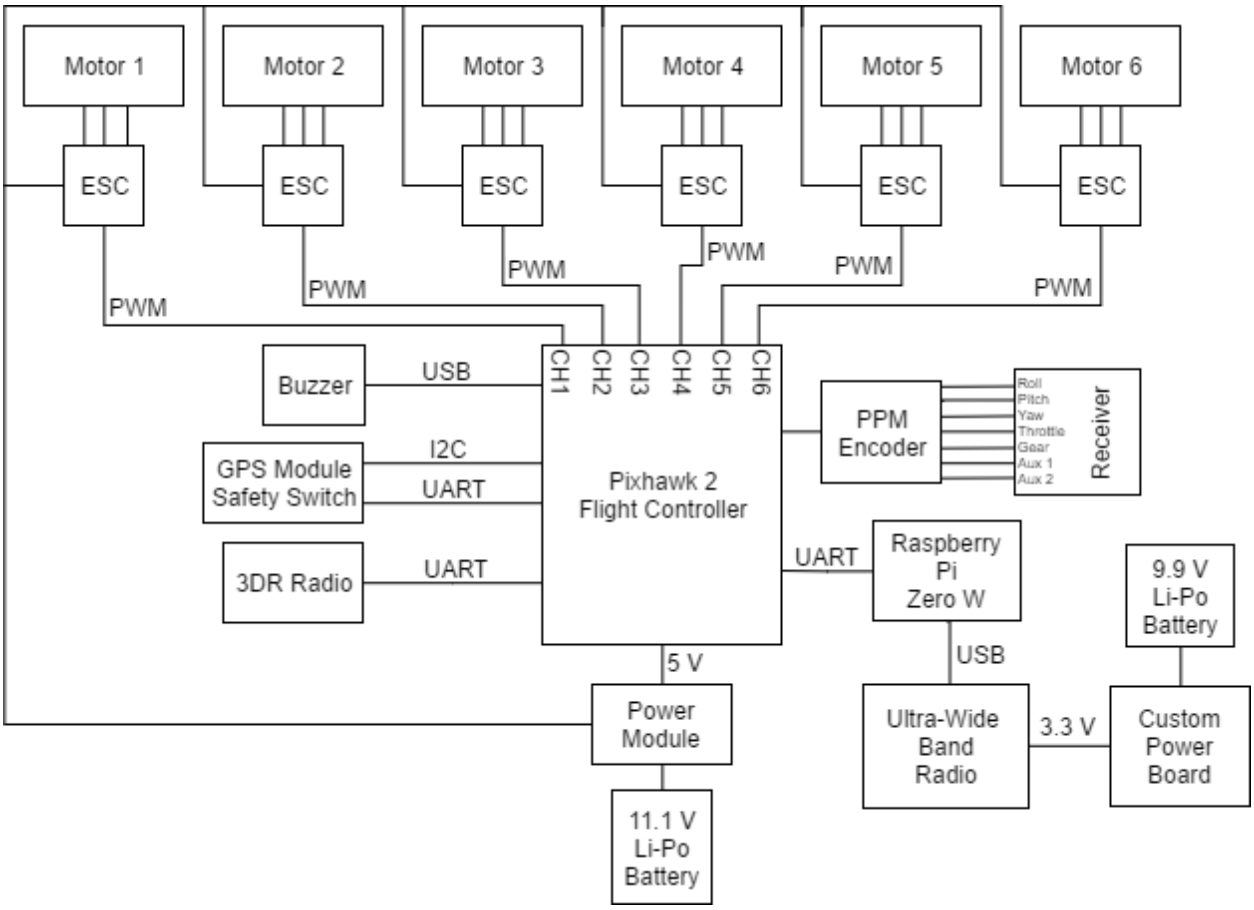


Figure 17.41: Final System Block Diagram

If at any point the code is ended or interrupted by the user, the system joins all threads, disconnects from the vehicle, and gracefully exits.

17. SYSTEM INTEGRATION (KHAI)

Figure 17.41 represents the final system after integrating all components onto the DJI F550 Hexacopter. Most significantly, the addition of the Raspberry Pi Zero Wireless and UWB modules are finally communicating with the Pixhawk 2 flight controller. From the block diagram, the UWB are powered by a separate power supply than from the flight controller. In addition, the raspberry pi is communicating to the controller through UART while the UWB is through serial. Altogether, the load on the hexacopter sums up to 4 pounds.

Furthermore, based on the system architecture, the Raspberry Pi Zero Wireless executes its onboard script to pull data from the UWB module. The information is then transmitted via UDP/IP to the follower UAS. Through the process of filtering and control system, the follower

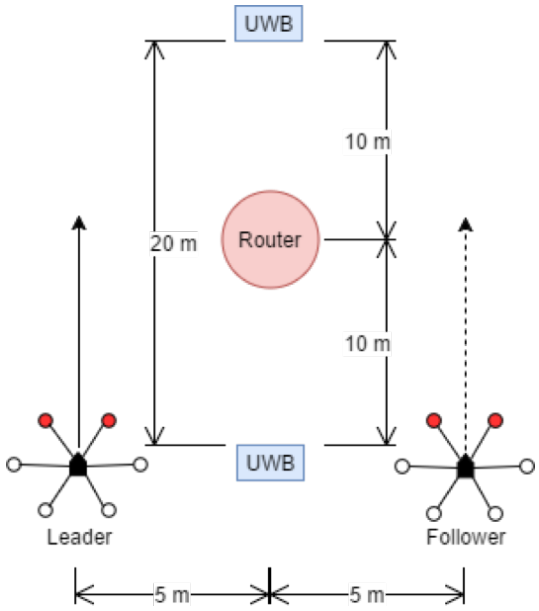


Figure 18.42: Experimental Setup.

sends commands from the raspberry pi to the Pixhawk 2 to perform the actions of matching its leader.

18. EXPERIMENTAL RESULTS (RAHUL)

The system was tested in an open field with the set of two UAS. The external UWB modules were placed on ladders at a distance 20m apart. A UAS was placed on either side of the ladder, with the forward rotors placed parallel to the line between each module. Figure 18.42 depicts the experimental setup of the system.

The system was then tested as follows:

1) Begin running leader data acquisition and transmission script.
2) Allow digital IIR filter to settle, and take off leader to 10m altitude.
3) Take off follower to 10m altitude, and subsequently begin execution of follower script.
4) After position filter settles on UWB module, fly leader UAS forward to specific 1-D location away from zero-positioned UWB module.
5) Broadcast leader position as x_{des} to follower via UDP socket.
6) Observe response of follower to leader position.

In testing, the proportional gain was increased to 0.005, however the system did not fully settle, and began oscillating about an incorrect location. It was determined that this incorrect

response was a result of flawed data being streamed from the leader vehicle. In a second test, the gain of the proportional controller was set to a value of 0.0005, and the derivative controller was set to a value of 0. In this test, the follower system responded to the motion of the leader, with an approximately 30s settling time.

19. FINAL BUDGET VS. COST ESTIMATES

The cost for funding this senior design project was estimated at approximately $2055. Since the software tools stated in Table 3 are free and available online, all costs are based on hardware components, itemized in Table 3. Table 11.22 shows a final breakdown of costs of all hardware purchased for this project. The total cost increased to approximately $2915. This increase over the estimated cost was due to the purchase of additional UWB modules, as well as the purchase of two hexacopters for increased flexibility in testing.

Table 3: Estimated Cost Breakdown

Hardware			
Component	**Quantity**	**Cost**	**Provider**
PixHawk 2	2	$400.00	ProfiCNC
TREK 1000	1	$1000.00	Decawave
DJI Flamewheel F450	2	$500.00	DJI
PCB	3	$20.00	Oshpark
Xbee S2C Module	2	$50.00	Digi-Key
CP2102 to Serial Adapter	2	$5.00	SparkFun
Intel Edison Module	2	$80.00	Intel
Software			
Software		**Cost**	**Notes**
Eagle CAD		Free	Software for PCB design.
DroneKit API		Free	Platform for designing UAS software using high level language.
Ardupilot Github		Free	Open Source repository for developers using PixHawk controller.
Total		**$2055**	

20. CONCLUSION

As the methodologies and results above show, there is sufficient theory motivating the implementation of the system design, and the results present an initial success of the UAS formation control system. The trilateration methodologies presented result in a position measurement of each UAS with sub-10cm accuracy, with little to no erroneous data present. The designed

Table 4: True Cost Breakdown

Hardware			
Component	**Quantity**	**Cost**	**Provider**
PixHawk 2	2	$400.00	Jester's Drones
TREK 1000	1	$1600.00	Decawave
DJI Flamewheel F550	2	$760.00	DJI
PCB	3	$50.00	Oshpark
Raspberry Pi Zero W	2	$80.00	Amazon
Batteries	2	$25	Hobbytown USA
Software			
Software		Cost	Notes
Eagle CAD		Free	Software for PCB design.
DroneKit API		Free	Platform for designing UAS software using high level language.
Ardupilot Github		Free	Open Source repository for developers using PixHawk controller.
Total		**$2915**	

controller results in the positions of both leader and follower matching with a sub-20cm accuracy. Future work will involve the tuning of the controller to decrease the system time constant, and also explore methods of decreasing latency between leader and follower.

ACKNOWLEDGEMENTS

We would like to thank the following people and organizations for their technical and financial support throughout the course of this project:

- Fresno State - Office of Undergraduate Studies
- NASA and the California Space Grant Consortium
- Fresno State Unmanned Systems Research Team
- Dr. Gregory Kriehn
- Tom Pittenger - Intel Corporation

REFERENCES

[1] M. Egerstedt and X. Hu, "Formation constrained multi-agent control," *IEEE Trans. Robot. Autom.*, pp. 947–951, 2001.

[2] F. Bullo, J. Cortes, and S. Martinez, *Distributed Control of Robotic Networks*. Princeton University Press, 2009.

[3] N. Michael and V. Kumar, "Control of ensembles of aerial robots," *Proc. IEEE*, vol. 99, pp. 1587–1602, Sep. 2011.

[4] M. W. Mueller, M. Hammer, and R. D'Andrea, "Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadcopter state estimation," *IEEE Int. Conf. Robot. Autom.*, vol. 99, pp. 1587–1602, Sep. 2011.

[5] R. Kiem. Back to basics: The universal asynchronous receiver/transmitter. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/back-to-basics-the-universal-asynchronous-receiver-transmitter-uart/

[6] L. Darmon, M. McLaughlin, and D. Neirynck, "Scensor, designing the first commercial ieee 802.15.4a chip," 2012.

[7] A. Carlson and P. Crilly, *Communication Systems*. Boston: McGraw-Hill Higher Education, 2010.

[8] D. Sundararajan, *Digital Signal Processing - Theory and Practice*. World Scientific Publishing Co. Pte. Ltd., 2003.

[9] N. Mohan, *Power Electronics*. John Wiley & Sons, Inc., 2012.

[10] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[11] G. Hoffmann, S. Waslander, and C. Tomlin, "Quadrotor helicopter trajectory tracking control," in *AIAA guidance, navigation and control conference and exhibit*, 2008, p. 7410.

[12] P. S. Foundation. Socket - low-level networking interface. [Online]. Available: https://docs.python.org/2/library/socket.html

FRESNO STATE
Discovery. Diversity. Distinction.

M E M O R A N D U M

June 9, 2016

Kai Chang
Dr. Gregory Kriehn

RE: Undergraduate Research Grant

I am pleased to inform you that your proposal Hexacopter Swarm Formation System submitted to the 2016-2017 Undergraduate Research Grant program has been funded.

For your project, $1000 is being made available to you to conduct your research. Grant funds can be used to cover project-related software, equipment, materials, supplies, travel, data collection and analysis, and other project-related expenses. No more than $500 can be used to pay you (as a student assistant). Funds may not be used for faculty compensation of any kind. You may begin spending on your project July 1st. You will need to work with your department office on any expenditures.

You are expected to work closely with your faculty mentor and complete your project by the end of the Spring 2017 semester or by the date specified in your proposal. You must also submit a final report. This can take the form of a publication, a presentation at a conference, a poster presentation on or off campus, a document submitted to my office, etc. Publications and presentations should recognize funding from the Undergraduate Research Grant program. If you would like to use a logo on a poster or in a PowerPoint presentation, we will make one available to you. As your project concludes, please let me know the form of your report.

The use of people and animals in research is federally regulated and you may need approval before proceeding with your research (Committee on the Protection of Human Subjects or Animal Care and Use Committee). Your faculty mentor can help you determine whether or not you need such approval and assist you with the approval process.

Congratulations and best wishes for a productive and successful project.

Xuanning Fu
Dean of Undergraduate Studies

Office of the Dean of Undergraduate Studies
California State University, Fresno • Henry Madden Library, Rm 4160
5200 North Barton Avenue M/S ML 54 • Fresno, California 93740-8014
P 559.278.4468 **F** 559.278.8340 ***www.FresnoState.edu*** THE CALIFORNIA STATE UNIVERSITY

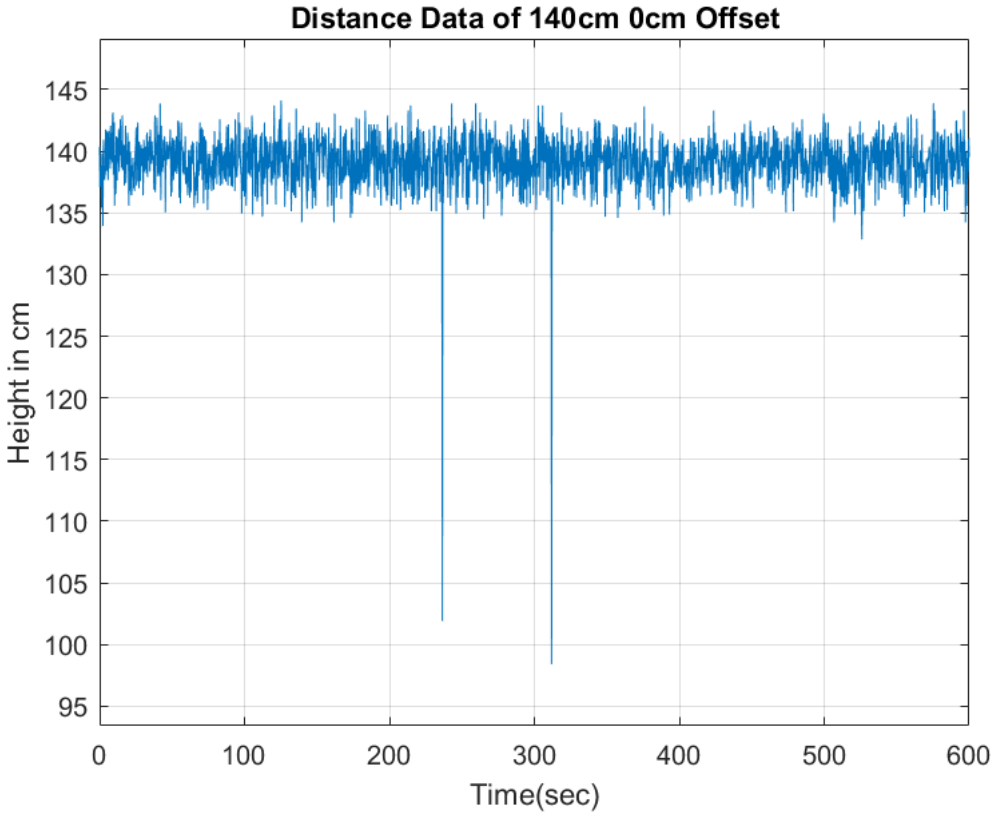
Distance Data Processing

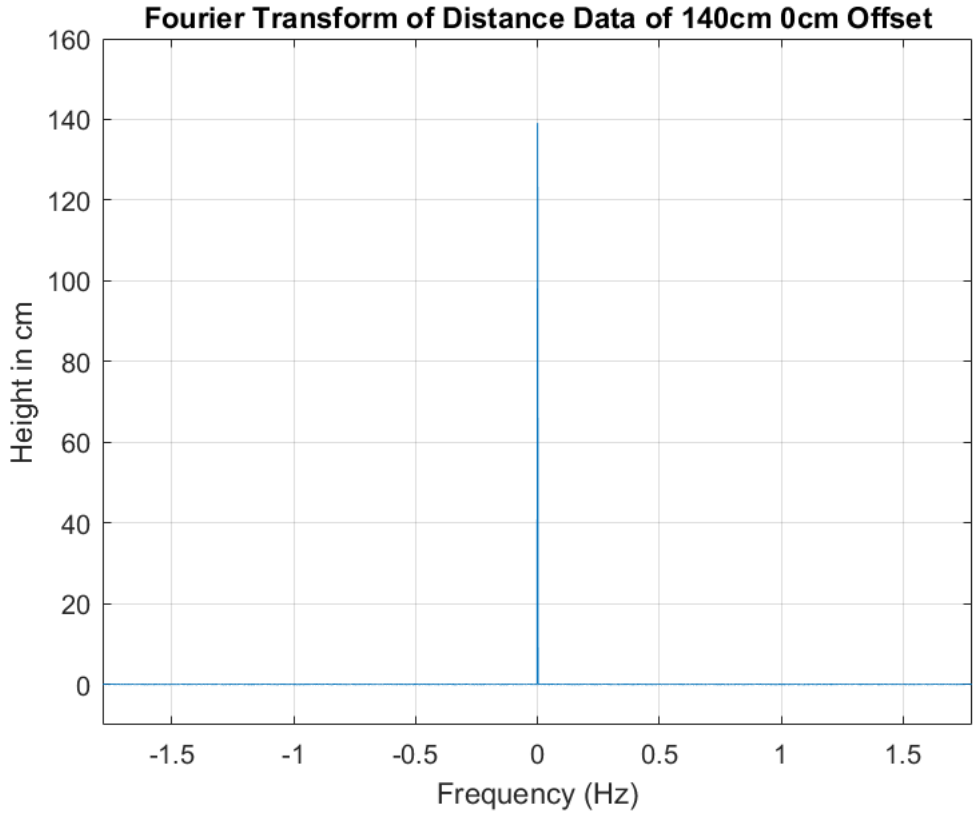
Table of Contents

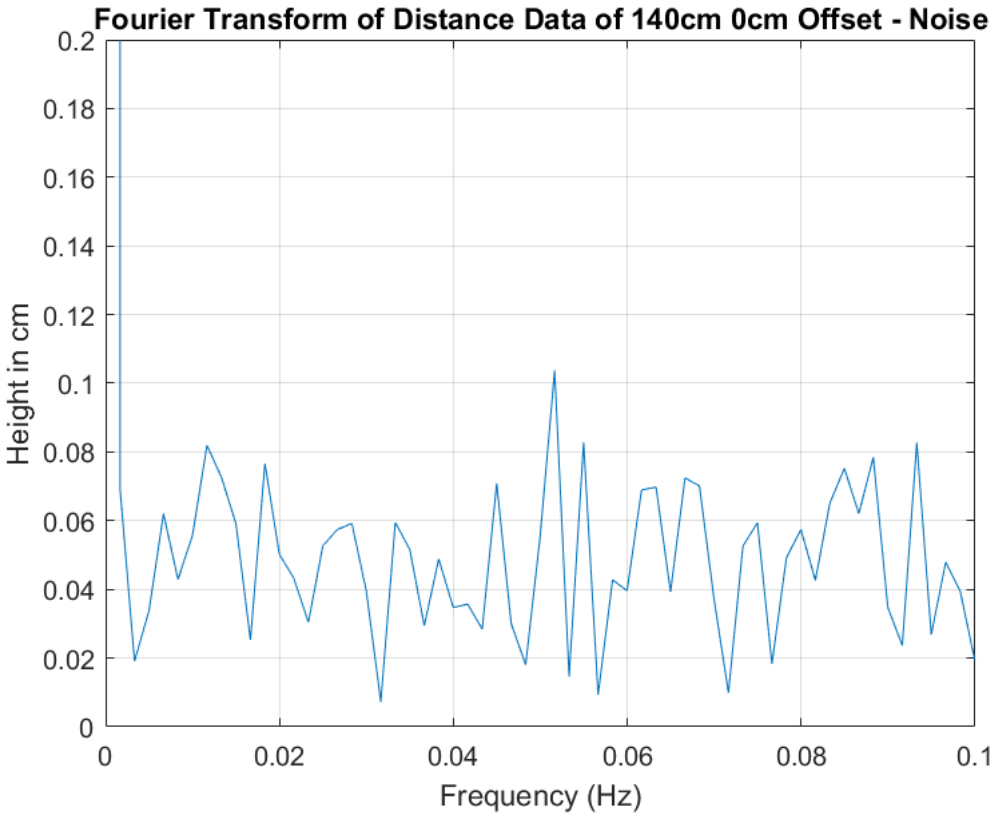
Static 0 cm offset

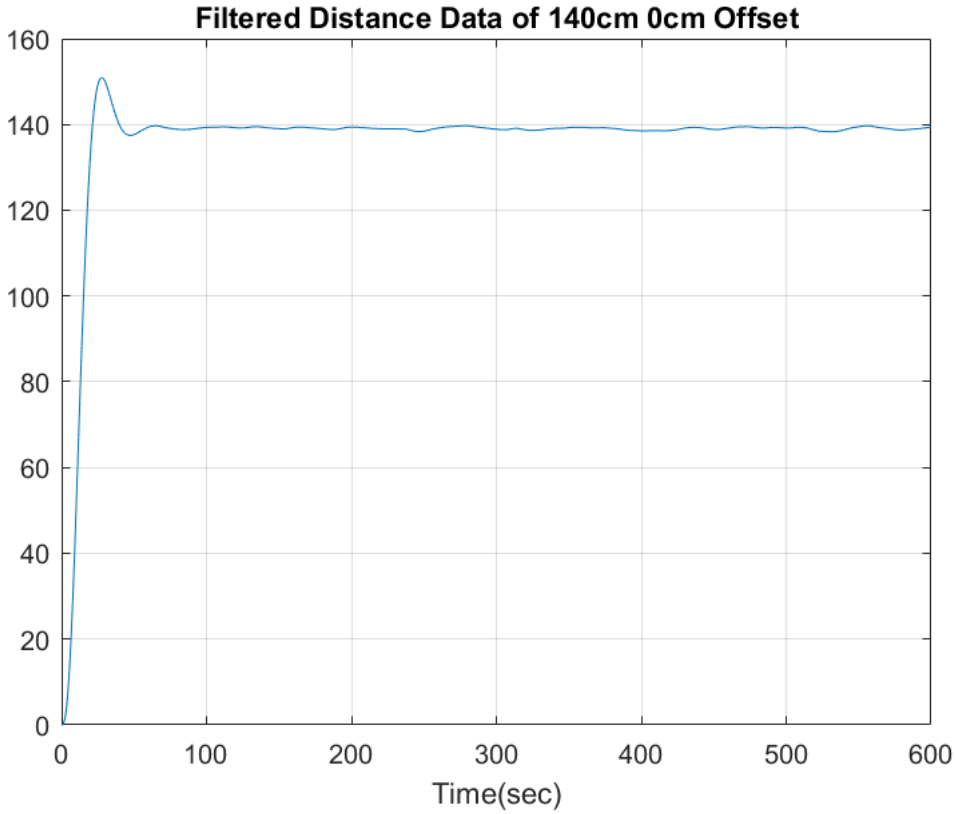
```
clear;
clc;

fileID1 = fopen('C:\Users\Reece\Documents\Senior_Design
\trilateration_processing\os_0cm__10min_perp.txt','r');
h1 = fscanf(fileID1,'%f',[1 Inf]);
n = length(h1);
t = linspace(0,10*60,n);

figure(1)
plot(t,h1)
hold on
axis([0 600 min(h1)-5 max(h1)+5])
grid on
title('Distance Data of 140cm 0cm Offset')
xlabel('Time(sec)')
ylabel('Height in cm')

sampling = t(2)-t(1);
N = length(t);
f = (-N/2:N/2 -1).*1/(N*sampling);
Xf = fftshift(fft(h1,N)/N);

figure(2)
plot(f,abs(Xf))
xlabel('Frequency (Hz)')
ylabel('Height in cm')
grid on
axis([min(f) max(f) -10 160])
title('Fourier Transform of Distance Data of 140cm 0cm Offset')

figure(22)
plot(f,abs(Xf))
xlabel('Frequency (Hz)')
ylabel('Height in cm')
grid on;
axis([0 0.1 0 0.2])
```

```
title('Fourier Transform of Distance Data of 140cm 0cm Offset -
 Noise')

fc = 500;
sampling = 0.00001;
[BB, AA] = butter(3, 2*pi*(fc/(1/(sampling/2))), 'low');
XTfilter = filter(BB, AA, h1);

figure(3)
plot(t,XTfilter)
title('Filtered Distance Data of 140cm 0cm Offset')
xlabel('Time(sec)')
grid on;
```




Fourier Transform of Distance Data of 140cm 0cm Offset
Height in cm
Frequency (Hz)
160
140
120
100
80
60
40
20
0
-1.5
-1
-0.5
0
0.5
1
1.5

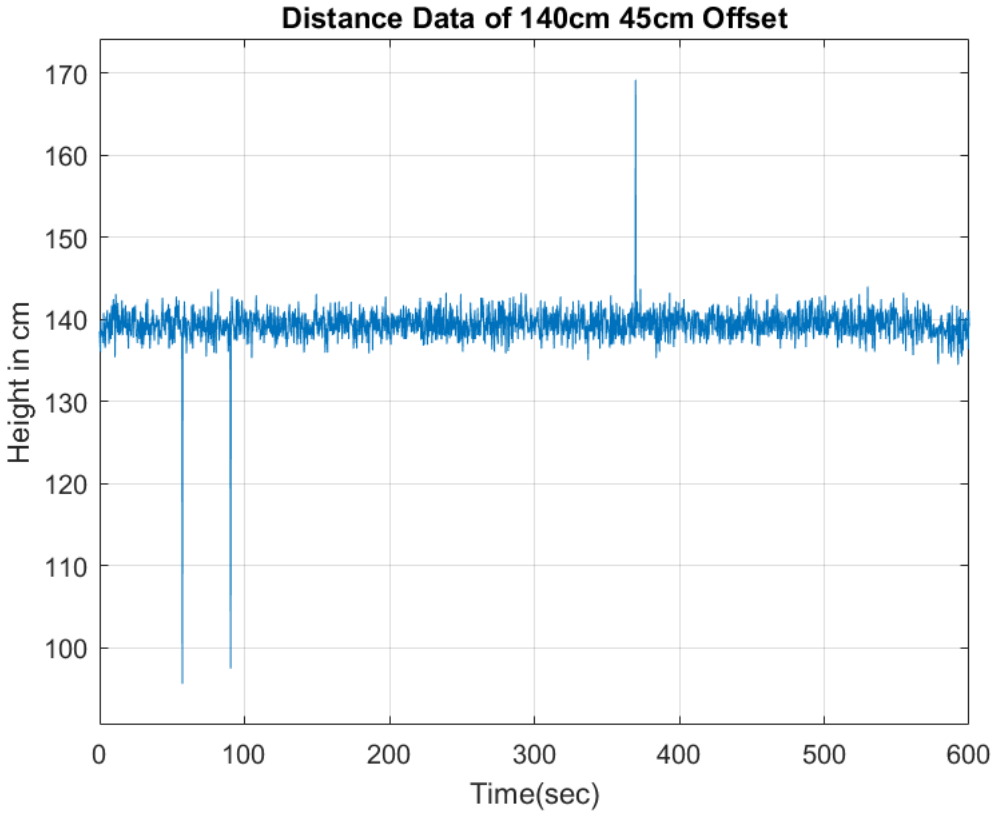

Fourier Transform of Distance Data of 140cm 0cm Offset - Noise
Height in cm
Frequency (Hz)
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
0.02
0.04
0.06
0.08
0.1

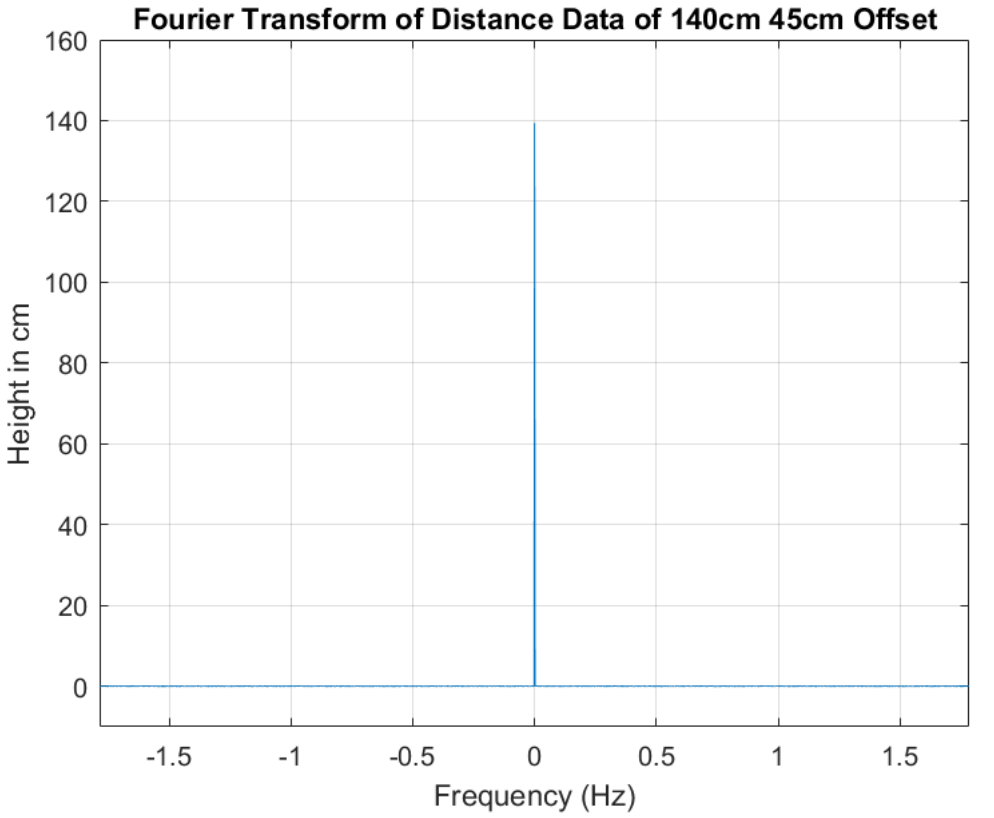


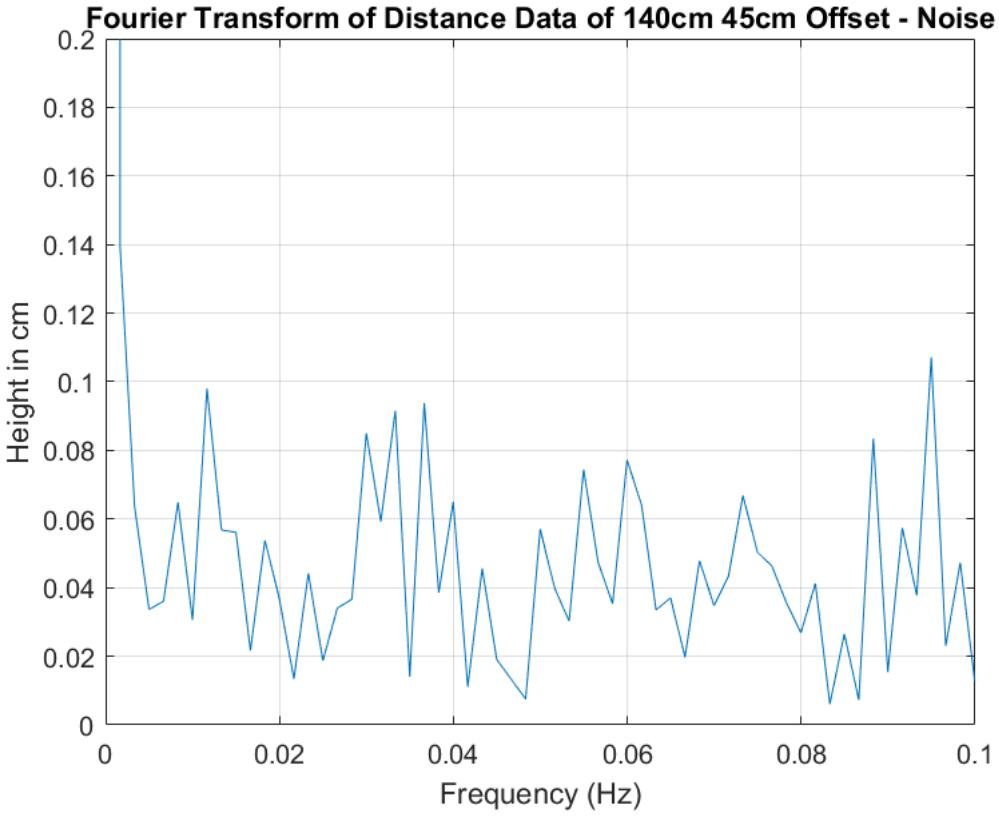
Static 45 cm offset

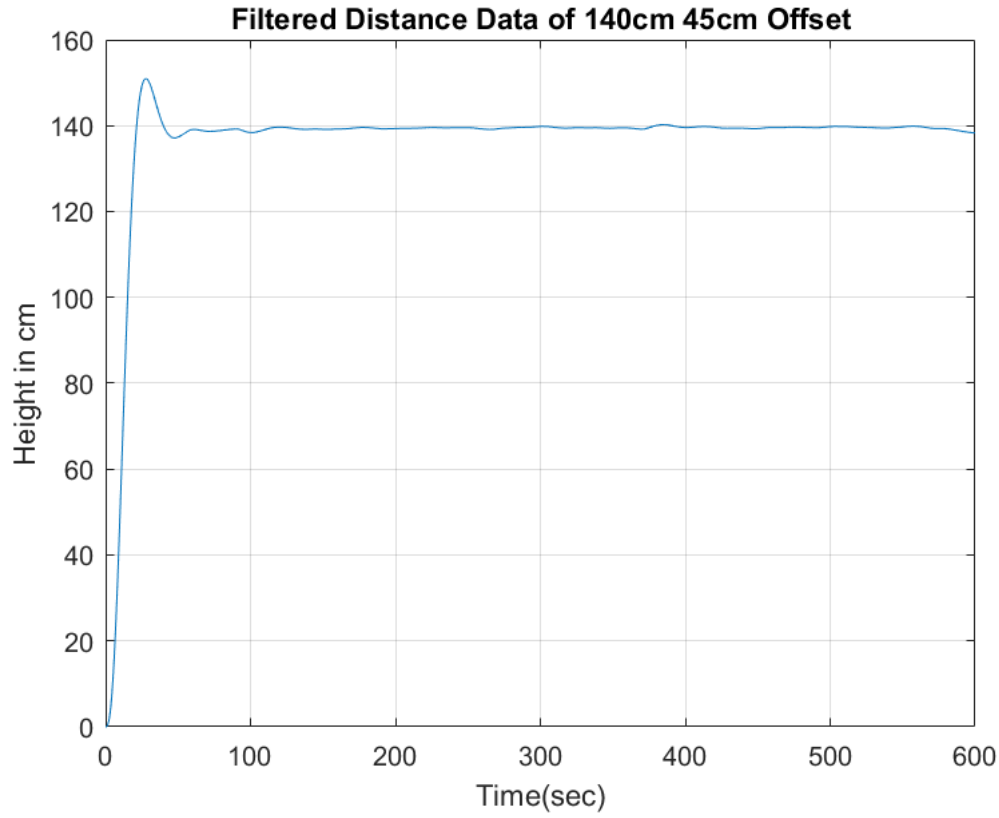
```
clear;
clc;

fileID1 = fopen('C:\Users\Reece\Documents\Senior_Design
\trilateration_processing\os_45cm__10min_perp.txt','r');
h1 = fscanf(fileID1,'%f',[1 Inf]);
n = length(h1);
t = linspace(0,10*60,n);

figure(4)
plot(t,h1)
hold on
axis([0 600 min(h1)-5 max(h1)+5])
grid on
title('Distance Data of 140cm 45cm Offset')
xlabel('Time(sec)')
ylabel('Height in cm')

sampling = t(2)-t(1);
N = length(t);
f = (-N/2:N/2 -1).*1/(N*sampling);
Xf = fftshift(fft(h1,N)/N);
```

```
figure(5)
plot(f,abs(Xf))
xlabel('Frequency (Hz)')
ylabel('Height in cm')
grid on
title('Fourier Transform of Distance Data of 140cm 45cm Offset')
axis([min(f) max(f) -10 160])

figure(52)
plot(f,abs(Xf))
xlabel('Frequency (Hz)')
ylabel('Height in cm')
grid on
title('Fourier Transform of Distance Data of 140cm 45cm Offset -
 Noise')
axis([0 0.1 0 0.2])

fc = 500;
sampling = 0.00001;
[BB, AA] = butter(3, 2*pi*(fc/(1/(sampling/2))), 'low');
XTfilter = filter(BB, AA, h1);

figure(6)
plot(t,XTfilter)
title('Filtered Distance Data of 140cm 45cm Offset')
xlabel('Time(sec)')
ylabel('Height in cm')
grid on;
```


Distance Data of 140cm 45cm Offset
Height in cm
170
160
150
140
130
120
110
100
0
100
200
300
400
500
600
Time(sec)


Fourier Transform of Distance Data of 140cm 45cm Offset
Height in cm
160
140
120
100
80
60
40
20
0
-1.5
-1
-0.5
0
0.5
1
1.5
Frequency (Hz)

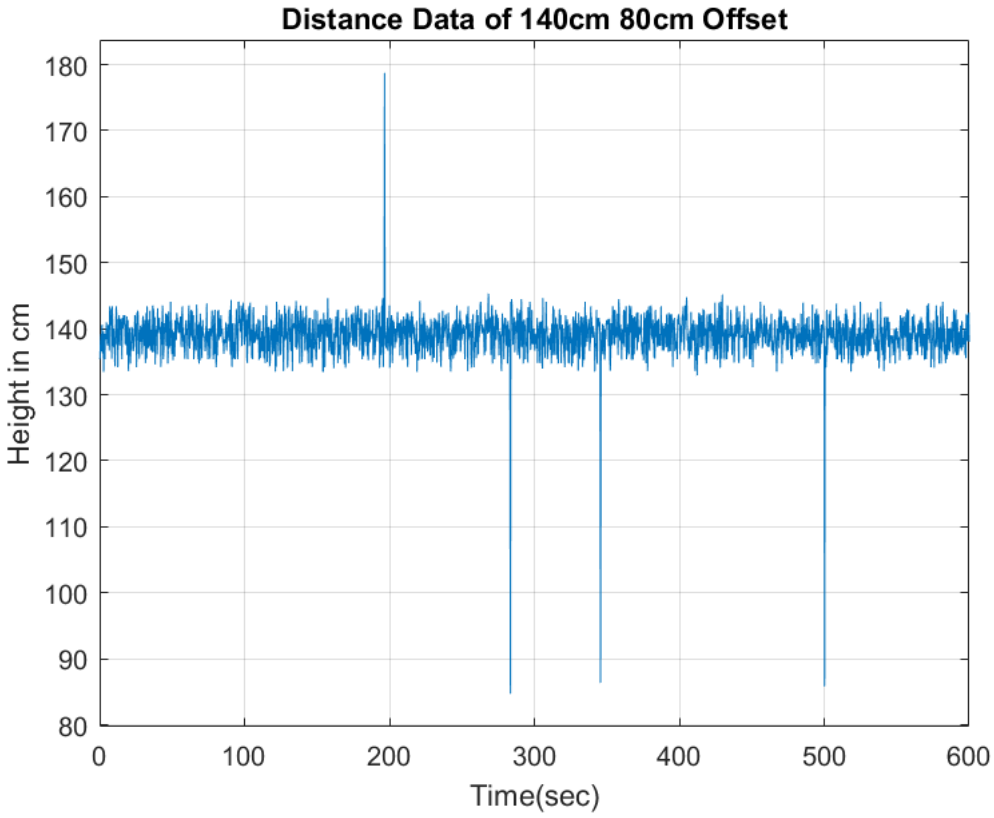

Fourier Transform of Distance Data of 140cm 45cm Offset - Noise
Height in cm
Frequency (Hz)
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2

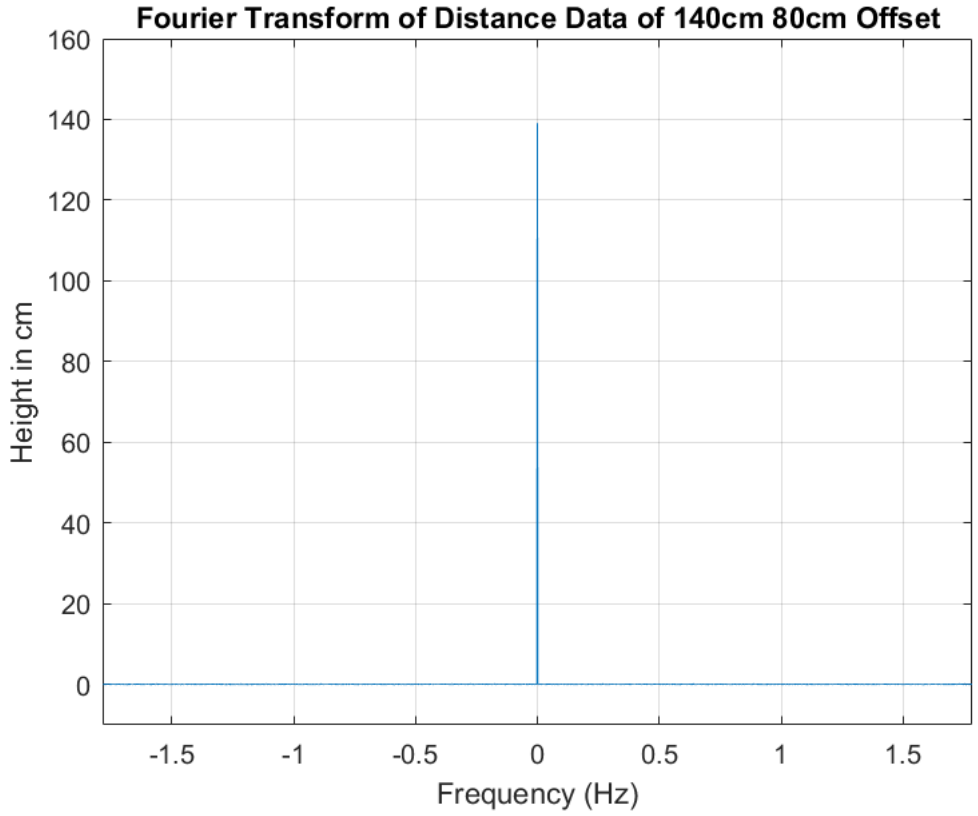

Filtered Distance Data of 140cm 45cm Offset
Height in cm
Time(sec)
0
20
40
60
80
100
120
140
160
0
100
200
300
400
500
600

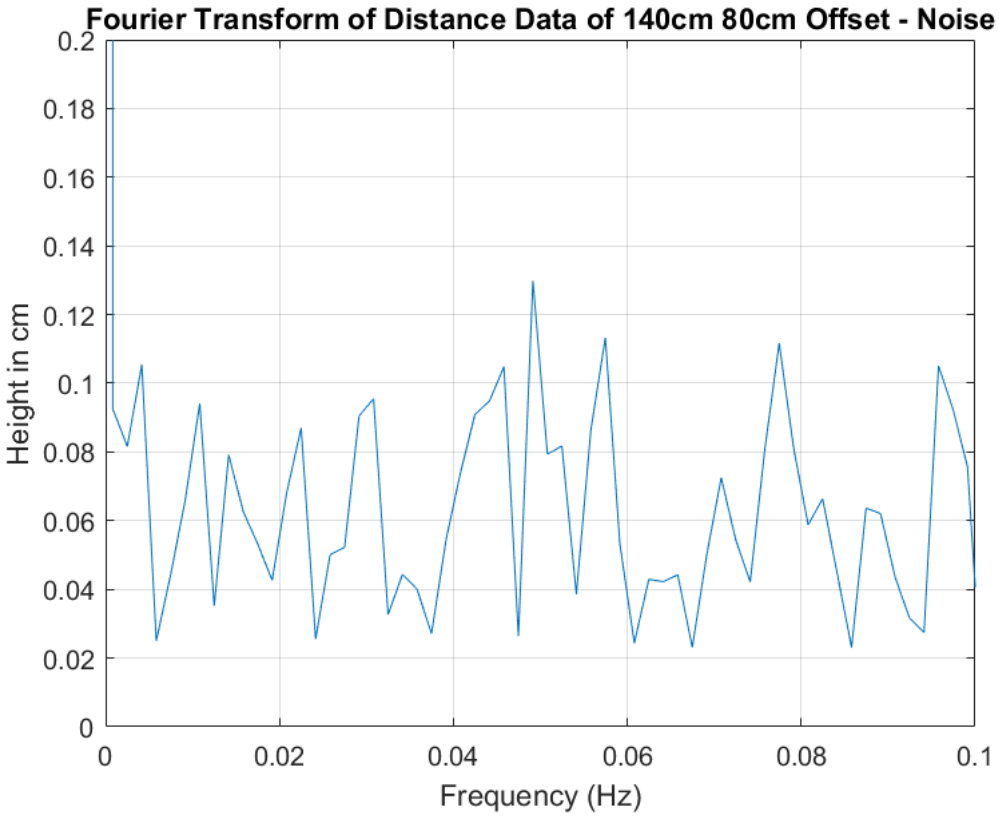
Static 80 cm offset

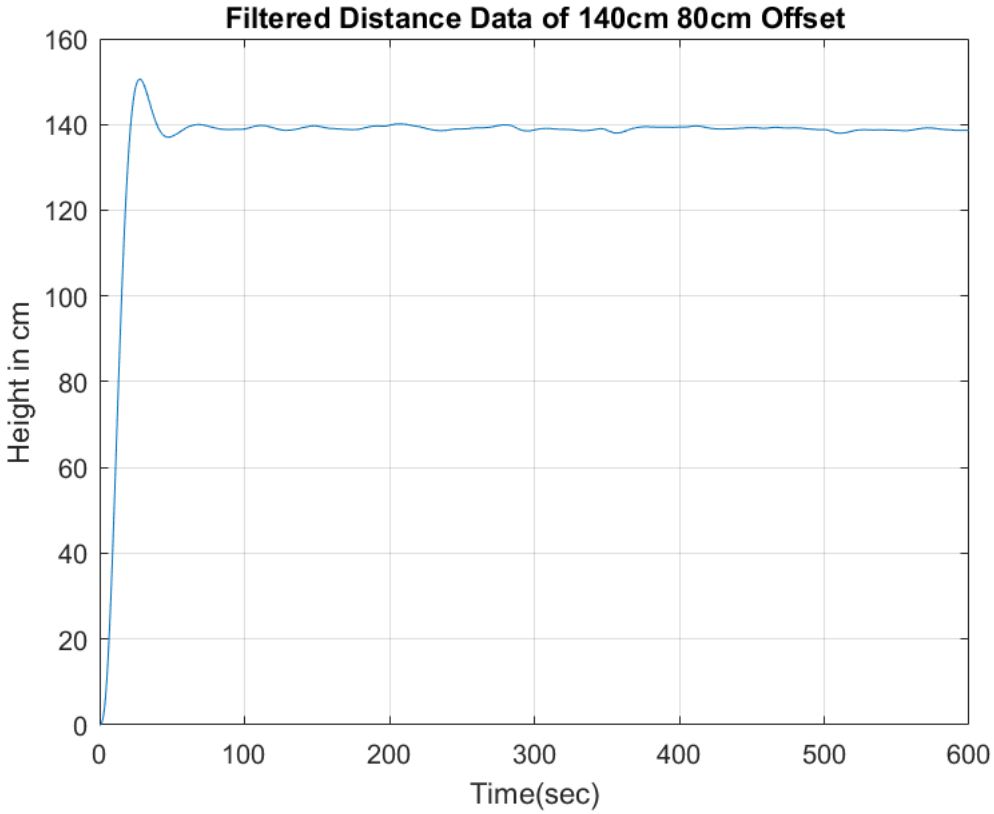
```
clear;
clc;

fileID1 = fopen('C:\Users\Reece\Documents\Senior_Design
\trilateration_processing\os_80cm__10min_perp.txt','r');
h1 = fscanf(fileID1,'%f',[1 Inf]);
n = length(h1);
t = linspace(0,10*60,n);

figure(7)
plot(t,h1)
hold on
axis([0 600 min(h1)-5 max(h1)+5])
grid on
title('Distance Data of 140cm 80cm Offset')
xlabel('Time(sec)')
ylabel('Height in cm')

sampling = t(2)-t(1);
N = length(t);
f = (-N/2:N/2 -1).*1/(N*sampling);
Xf = fftshift(fft(h1,N)/N);

figure(8)
plot(f,abs(Xf))
xlabel('Frequency (Hz)')
ylabel('Height in cm')
grid on
title('Fourier Transform of Distance Data of 140cm 80cm Offset')
axis([min(f) max(f) -10 160])


figure(82)
plot(f,abs(Xf))
xlabel('Frequency (Hz)')
ylabel('Height in cm')
grid on
title('Fourier Transform of Distance Data of 140cm 80cm Offset -
 Noise')
axis([0 0.1 0 0.2])


fc = 500;
sampling = 0.00001;
[BB, AA] = butter(3, 2*pi*(fc/(1/(sampling/2))), 'low');
XTfilter = filter(BB, AA, h1);

figure(9)
plot(t,XTfilter)
title('Filtered Distance Data of 140cm 80cm Offset')
xlabel('Time(sec)')
```

```
ylabel('Height in cm')
grid on;
```




Fourier Transform of Distance Data of 140cm 80cm Offset
Height in cm
Frequency (Hz)
160
140
120
100
80
60
40
20
0
-1.5
-1
-0.5
0
0.5
1
1.5


Fourier Transform of Distance Data of 140cm 80cm Offset - Noise
Height in cm
Frequency (Hz)
0.2
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
0.02
0.04
0.06
0.08
0.1

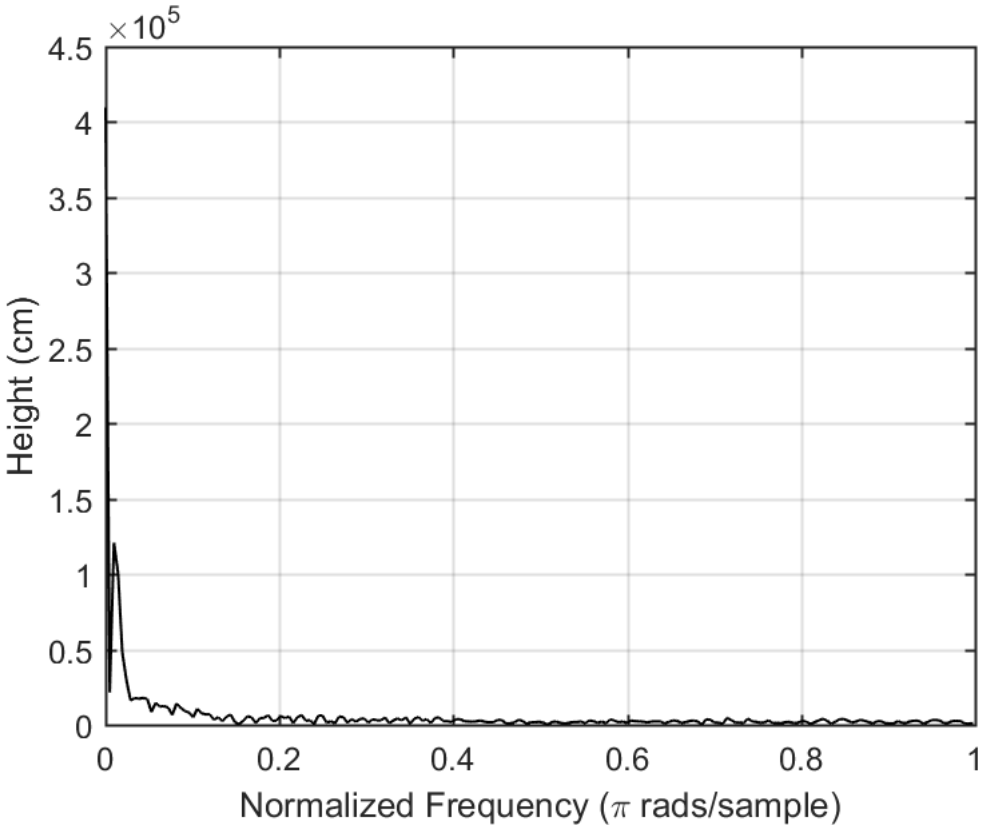


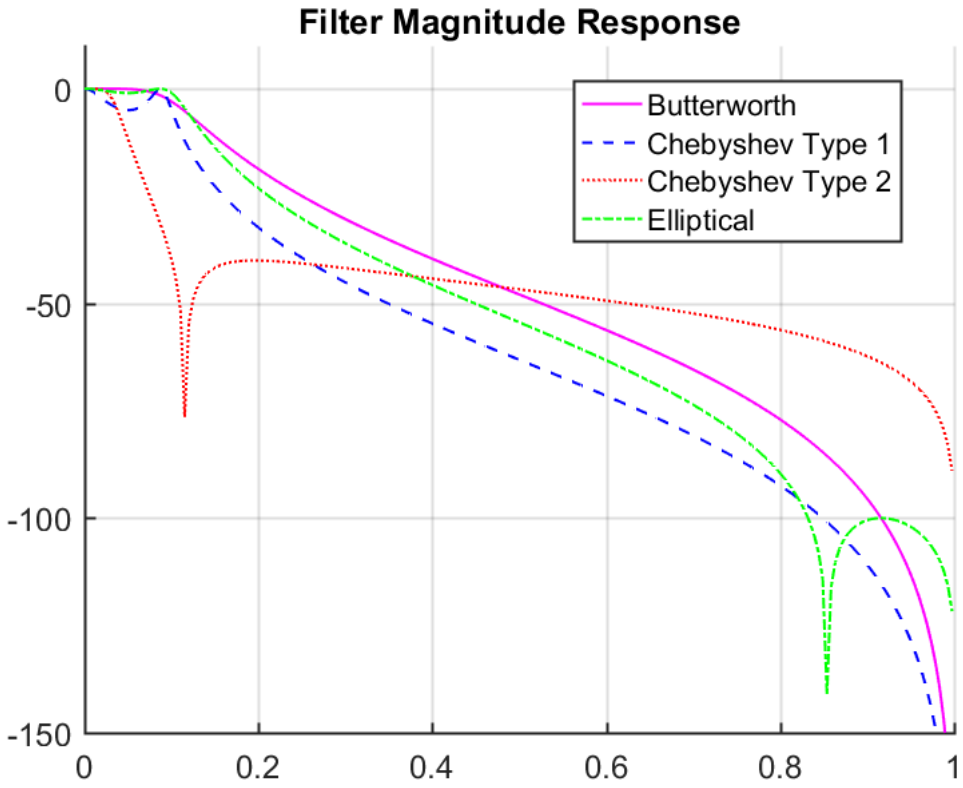
Dyanmic Data Collection

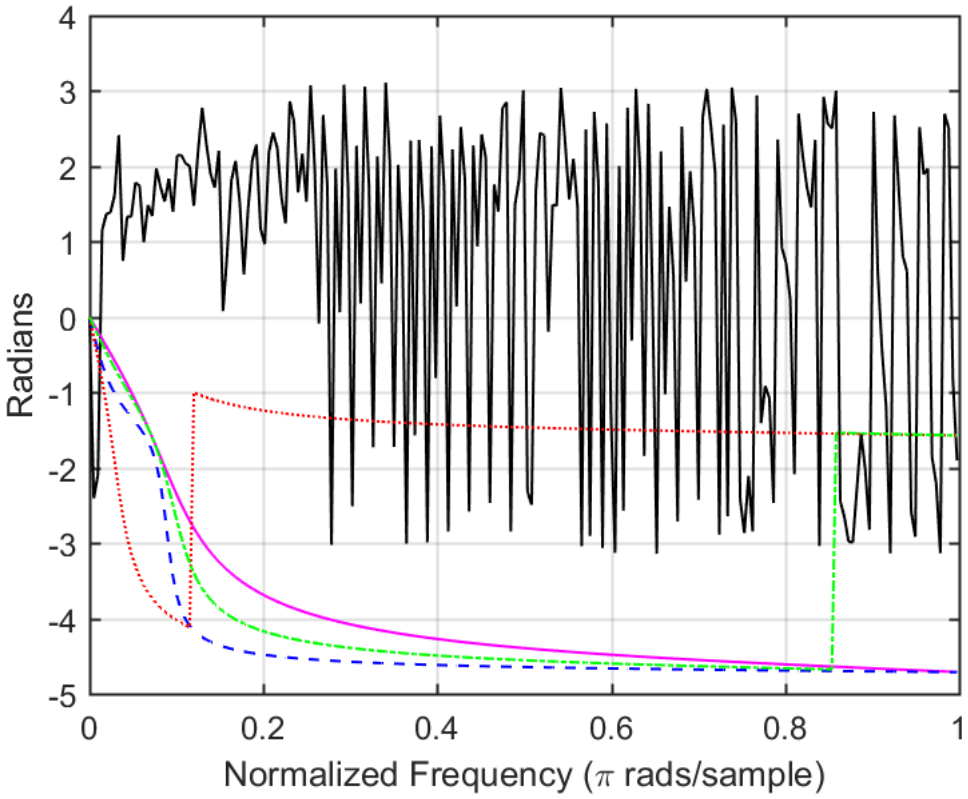
```
clear;
clc;

%o fileID1 = fopen('C:\Users\Reece\Documents\Senior_Design
\trilateration_processing\os_80cm__10min_perp.txt','r');

 fileID1 = fopen('C:\Users\Reece\Documents\Senior_Design
\trilateration_processing\dynamic_20m_2min_2m-offset.txt','r');

% fileID1 = fopen('C:\Users\Reece\Documents\Senior_Design
\trilateration_processing\dynamic_20m_2min_no_offset.txt','r');

% x1 = Raw Position Data
x1 = fscanf(fileID1,'%f',[1 Inf]);
x1_len = length(x1);
% n is the amount of Raw Data points
n1 = 0:1:x1_len-1;
% X1 = DFT of the Raw Position Data (n - pt is defined by the n above
X1 = fft(x1,x1_len);

% N = Order of Filter Transfer Function
N = 3;
% Wp = Passband Frequency/ Cut Off Frequency
```

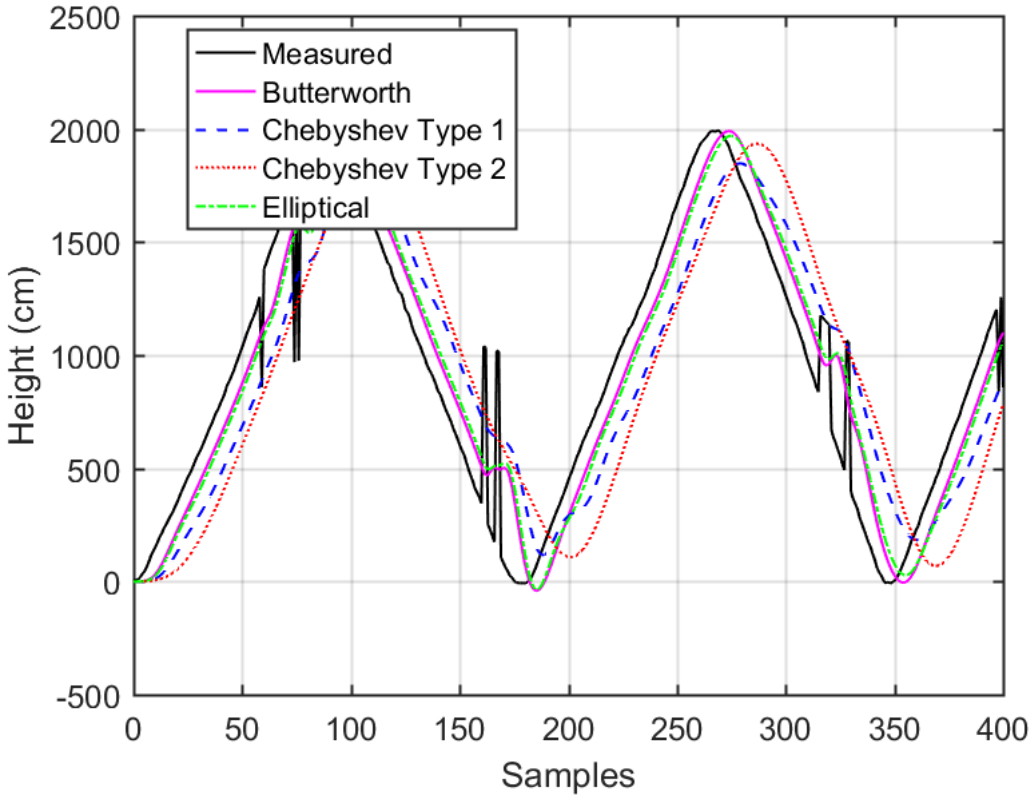
```
Wp = 0.1;

figure(111);
% Plot of the Raw Position Data,
% The data is discrete but to view the data more easily it has been
 plotted
% as a line due to the amount of data points.
plot(n1, x1,'k');
xlabel('Samples');
ylabel('Height (cm)');
grid on;

% Frequency Normalization
num_bins = length(X1);
w = [0:1/(num_bins/2-1):1];

figure(222);
% Plot of the Frequency content of the Raw Position Data - Magnitude
% Response
plot(w, abs(X1(1:num_bins/2)),'k');
grid on;
xlabel('Normalized Frequency (\pi rads/sample)');
ylabel('Height (cm)');
set(gca,'FontSize',12);
set(findobj(gcf,'LineWidth',0.5),'LineWidth',1);

figure(333);
% Plot of the Frequency content of the Raw Position Data - Phase
 Response
%plot(w, unwrap(angle(X1(1:num_bins/2))),'b');
plot(w, angle(X1(1:num_bins/2)),'k');
grid on;
xlabel('Normalized Frequency (\pi rads/sample)');
ylabel('Radians');

% Butterworth Filter
[b_butter, a_butter] = butter(N, Wp, 'low');
H_butter = freqz(b_butter, a_butter, floor(num_bins/2));

% Chebyshev Type 1 Filter
[b_cheby1, a_cheby1] = cheby1(N, 5, Wp, 'low');
H_cheby1 = freqz(b_cheby1, a_cheby1, floor(num_bins/2));

% Chebyshev Type 2 Filter
[b_cheby2, a_cheby2] = cheby2(N, 40, Wp, 'low');
H_cheby2 = freqz(b_cheby2, a_cheby2, floor(num_bins/2));

% Ellipitical Filter
[b_ellip, a_ellip] = ellip(N, 1, 100, Wp, 'low');
H_ellip = freqz(b_ellip, a_ellip, floor(num_bins/2));

figure(2222);
% Addition of the Filter Transfer Functions - Magnitude Respone
hold on;
```

```
plot(w, 20*log10(abs(H_butter)),'-m');
plot(w, 20*log10(abs(H_cheby1)),'--b');
plot(w, 20*log10(abs(H_cheby2)),':r');
plot(w, 20*log10(abs(H_ellip)),'-.g');
hold off;
grid on;
ylim([-150 10])
set(gca,'FontSize',12);
set(findobj(gcf,'LineWidth',0.5),'LineWidth',1);
title('Filter Magnitude Response')
legend('Butterworth','Chebyshev Type 1','Chebyshev Type 2', ...
    'Elliptical','Location','Best')

figure(333);
% Addition of the Filter Transfer Functions - Phase Response
hold on;
plot(w, unwrap(angle(H_butter)),'-m');
plot(w, unwrap(angle(H_cheby1)),'--b');
plot(w, unwrap(angle(H_cheby2)),':r');
plot(w, unwrap(angle(H_ellip)),'-.g');
hold off;
set(gca,'FontSize',12);
set(findobj(gcf,'LineWidth',0.5),'LineWidth',1)

% Filtering of the Position Data
x1_filter_butter = filter(b_butter, a_butter, x1);
x1_filter_cheby1 = filter(b_cheby1, a_cheby1, x1);
x1_filter_cheby2 = filter(b_cheby2, a_cheby2, x1);
x1_filter_ellip = filter(b_ellip, a_ellip, x1);


figure(111)
% Addition of the Filter output to the figure, to compare the response
 of
% the filters
hold on;
plot(n1, x1_filter_butter,'-m');
plot(n1, x1_filter_cheby1,'--b');
plot(n1, x1_filter_cheby2,':r');
plot(n1, x1_filter_ellip,'-.g');
legend('Measured','Butterworth','Chebyshev Type 1','Chebyshev Type
 2', ...
    'Elliptical','Location','Best')
hold off;
set(gca,'FontSize',12);
set(findobj(gcf,'LineWidth',0.5),'LineWidth',1)
xlim([0 400])

Warning: Integer operands are required for colon operator when used as
 index
Warning: Integer operands are required for colon operator when used as
 index
```


×10^5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Height (cm)
0
0.2
0.4
0.6
0.8
1
Normalized Frequency (π rads/sample)


Filter Magnitude Response
Butterworth
Chebyshev Type 1
Chebyshev Type 2
Elliptical
0
-50
-100
-150
0
0.2
0.4
0.6
0.8
1


Radians
Normalized Frequency (π rads/sample)


Measured
Butterworth
Chebyshev Type 1
Chebyshev Type 2
Elliptical
Height (cm)
Samples

Published with MATLAB® R2016a

```python
################################################################################
#   follower_control_dynamic.py
#   Rahul Nunna, 2017
#   Main script for follower.
################################################################################

# import dronekit_sitl
import time
import serial
import re
import threading
import socket
import sys
from dronekit import connect, VehicleMode
from pymavlink import mavutil
from velocity_fns import send_body_ned_velocity, send_body_ned_velocity_logging
from uwb import uwb
from clientsocket import clientsocket

# print "Start simulator (SITL)"
# sitl = dronekit_sitl.start_default()
# connection_string = sitl.connection_string()


def control(client):
    lasterror = 0
    while True:
        myPos = radio.getRange()
        goal = float(client.receive())
        error = goal-myPos
        kp = 0.003
        kd = 0
        print "My current position:" + str(myPos)
        print "Desired position: " + str(goal)
        if (abs(error) > 20):
            prop = (error)*kp
            deriv = ((error-lasterror)/0.1)*kd
            speed = prop + deriv
            send_body_ned_velocity(vehicle, speed, 0, 0)
            # myPos = radio.getRange()
        else:
            send_body_ned_velocity(vehicle, 0, 0, 0)
        lasterror = error


# filename1 = "pos_gps" + time.strftime("%m_%d_%H%M") + ".txt"
# filename2 = "vel_imu" + time.strftime("%m_%d_%H%M") + ".txt"
# filename3 = "pos_uwb_filter" + time.strftime("%m_%d_%H%M") + ".txt"
# filename4 = "pos_uwb_raw" + time.strftime("%m_%d_%H%M") + ".txt"
# pos_file = open(filename1, 'w+')
# vel_file = open(filename2, 'w+')
# uwb_file = open(filename3, 'w+')
# uwb_raw = open(filename4, 'w+')
# pos_file.truncate()
# vel_file.truncate()
# uwb_file.truncate()
# uwb_raw.truncate()

try:
    host = ""
    port = 5456
    # Connect to the Vehicle.
```

```
63      print("Connecting to vehicle & initializing UWB & WiFi")
64      client = clientsocket(host, port)  # wifi init
65      print "connecting to vehicle"
66      vehicle = connect('/dev/ttyS0', wait_ready=True, baud=921600)  # vehicle init
67      print "Connecting to UWB radio"
68      radio = uwb(a=2000, port='/dev/ttyACM0')  # UWB init
69      print 'Connected. Starting to measure position...'
70      radiothread = threading.Thread(target=radio.range,
71                                     args=(False,))
72      radiothread.start()
73
74      # Get some vehicle attributes (state)
75      print "Get some vehicle attribute values:"
76      print " GPS: %s" % vehicle.gps_0
77      print " Battery: %s" % vehicle.battery
78      print " Last Heartbeat: %s" % vehicle.last_heartbeat
79      print " Is Armable?: %s" % vehicle.is_armable
80      print " System status: %s" % vehicle.system_status.state
81      print " Mode: %s \n" % vehicle.mode.name    # settable
82
83      t_end = time.time() + (10)
84      while time.time() < t_end:
85          print "Waiting for filter to settle..."
86          time.sleep(1)
87
88      while not vehicle.is_armable:
89          print " Is Armable?: %s" % vehicle.is_armable
90          time.sleep(1)
91
92      print " Is Armable?: %s" % vehicle.is_armable
93      vehicle.mode = VehicleMode("GUIDED")
94      # vehicle.armed = True
95
96      # while not vehicle.armed:
97      #     print " Waiting for arming..."
98      #     # print " Mode: %s" % vehicle.mode.name    # settable
99      #     time.sleep(1)
100
101     # while True:
102     #     pass
103
104     # print "Taking off!"
105     # height = 5
106     # vehicle.simple_takeoff(height)
107     # while True:
108     #     print " Altitude: ", vehicle.location.global_relative_frame.alt
109     #     if vehicle.location.global_relative_frame.alt >= height*0.95:
110     #         # Trigger just below target alt.
111     #         print "Reached target altitude"
112     #         break
113     #     time.sleep(1)
114
115     print 'Moving to desired position!'
116
117     # controlthread = threading.Thread(target=control, args=(client,))
118     # controlthread.start()
119
120     while True:
121         pass
122
123 except KeyboardInterrupt:
124     # pos_file.close()
```

```
125     # vel_file.close()
126     # uwb_file.close()
127     # uwb_raw.close()
128     radiothread.join()
129     controlthread.join()
130     vehicle.close()
131     sys.exit(0)
132 
133 # Shut down simulator
134 # sitl.stop()
135 
```

```
1  ##############################################################################
2  #   follower_control_dynamic.py
3  #   Rahul Nunna, 2017
4  #   Main script for leader.
5  ##############################################################################
6
7  from uwb import uwb
8  from serversocket import serversocket
9  import threading
10
11 host = "192.168.1.106"
12 port = 5456
13
14 print "Initializing UWB radio."
15 radio = uwb(a=2000, port='/dev/ttyACM1')  # UWB init
16 server = serversocket(host, port)
17 radiothread = threading.Thread(target=radio.range,
18                                 args=(True, server))
19 print "Calculating position..."
20 radiothread.start()
21
```

```
1  ##############################################################################
2  #   velocity_fns.py
3  #   Rahul Nunna, 2017
4  #   Custom dronekit velocity functions, create custom mavlink messages.
5  ##############################################################################
6
7  from dronekit import connect, VehicleMode
8  from pymavlink import mavutil
9  import time
10 import re
11
12
13 sleeptime = 0.1
14
15
16 def send_body_ned_velocity(vehicle, velocity_x, velocity_y, velocity_z):
17     """
18     Move vehicle in direction based on specified velocity vectors.
19     """
20     msg = vehicle.message_factory.set_position_target_local_ned_encode(
21         0,       # time_boot_ms (not used)
22         0, 0,    # target system, target component
23         # mavutil.mavlink.MAV_FRAME_BODY_NED,  # frame
24         mavutil.mavlink.MAV_FRAME_BODY_NED,  # frame
25         0b0000111111000111,  # type_mask (only speeds enabled)
26         0, 0, 0,  # x, y, z positions (not used)
27         velocity_x, velocity_y, velocity_z,  # x, y, z velocity in m/s
28         0, 0, 0,
29         # x, y, z acceleration (not supported yet, ignored in GCS_Mavlink)
30         0, 0)    # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)
31
32     # send command to vehicle on 5 Hz cycle
33     # print str(vehicle.location.global_relative_frame)
34     # print str(vehicle.velocity)
35     vehicle.send_mavlink(msg)
36     time.sleep(sleeptime)
37
38
39 def send_body_ned_velocity_logging(vehicle, velocity_x, velocity_y,
40                                    velocity_z, pos_file, vel_file):
41     """
42     Move vehicle in direction based on specified velocity vectors.
43     """
44     msg = vehicle.message_factory.set_position_target_local_ned_encode(
45         0,       # time_boot_ms (not used)
46         0, 0,    # target system, target component
47         mavutil.mavlink.MAV_FRAME_BODY_NED,  # frame
48         # mavlink.MAV_FRAME_LOCAL_NED, # frame
49         0b0000111111000111,  # type_mask (only speeds enabled)
50         0, 0, 0,  # x, y, z positions (not used)
51         velocity_x, velocity_y, velocity_z,  # x, y, z velocity in m/s
52         0, 0, 0,
53         # x, y, z acceleration (not supported yet, ignored in GCS_Mavlink)
54         0, 0)    # yaw, yaw_rate (not supported yet, ignored in GCS_Mavlink)
55
56     # send command to vehicle on 1 Hz cycle
57     vehicle.send_mavlink(msg)
58     latitude = re.search("lat=(.*?),",
59                          str(vehicle.location.global_relative_frame)).group(1)
60     longitude = re.search("lon=(.*),",
61                           str(vehicle.location.global_relative_frame)).group(1)
62     altitude = re.search("alt=(.*)",
```

```
63                          str(vehicle.location.global_relative_frame)).group(1)
64      vel_x = re.search("\[(.*?),", str(vehicle.velocity)).group(1)
65      vel_y = re.search(", (.*),", str(vehicle.velocity)).group(1)
66      vel_z = re.search(".+\, (.+)]", str(vehicle.velocity)).group(1)
67      pos_file.write(latitude + ',' + longitude + ',' + altitude + "\n")
68      vel_file.write(vel_x + ',' + vel_y + ',' + vel_z + "\n")
69      time.sleep(sleeptime)
70
```

```
################################################################################
#   uwb.py
#   Rahul Nunna, 2017
#   UWB class.
################################################################################

import serial
import math


class uwb:

    def __init__(self, a=10000, port='/dev/ttyACM0', mybaud=9600):
        self._n = 0
        self._rawDist = []
        self.filteredDist = []
        self._anchor1 = 0
        self._anchor2 = 0
        self._ser = serial.Serial(port, baudrate=mybaud)
        self.totalRange = a
        self._buffer = ''

    def _getRawDist(self):
        # while True:
        #     if '\n' in self._buffer:
        #         break
        #     else:
        #         self._buffer += self._ser.read(self._ser.inWaiting())
        #
        # line, self._buffer = self._buffer.split('\n')[-2:]
        line = self._ser.readline()
        # print line
        if line[:1] == 'm':
            # print line
            if line[:2] == 'mc':
                if len(line) == 65:
                    # print str(len(line)) + ": " + str(line)
                    self._anchor0 = int(line[6:14], 16)/10
                    self._anchor1 = int(line[15:23], 16)/10
                    height = (math.pow(self._anchor0, 2) -
                              math.pow(self._anchor1, 2) +
                              math.pow(self.totalRange, 2))/(2*self.totalRange)
                    self._rawDist.append(height)
                else:
                    self._getRawDist()
            else:
                self._getRawDist()
        else:
            self._getRawDist()

    def range(self, isLeader=False, server=''):
        while True:
            n = self._n
            x = self._rawDist
            y = self.filteredDist
            self._getRawDist()
            a = [1, -2.0651, 1.5200, -0.3861]
            b = [0.0086, 0.0258, 0.0258, 0.0086]
            if n >= 3:
                y.append(b[0]*x[n] + b[1]*x[n-1] + b[2]*x[n-2] + b[3]*x[n-3] -
                         a[1]*y[n-1] - a[2]*y[n-2] - a[3]*y[n-3])
            else:
```

```
63                  y.append(x[n])
64              n = n+1
65              self._n = n
66              self._rawDist = x
67              self.filteredDist = y
68              # rawfile.write(str(x[-1]) + '\n')
69              # textfile.write(str(y[-1]) + '\n')
70              if isLeader:
71                  server.send(str(y[-1]) + '\n')
72
73      def getRange(self):
74          return self.filteredDist[-1]
75
```

```
1  ##############################################################################
2  #   serversocket.py
3  #   Rahul Nunna, 2017
4  #   Server socket class, for follower.
5  ##############################################################################
6
7  import socket
8
9
10 class clientsocket:
11     def __init__(self, host, port=5454):
12         self._client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
13         self._client.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
14         self._client.bind((host, port))
15
16     def receive(self):
17         return self._client.recv(1024)
18
```

```
1  ##############################################################################
2  #   serversocket.py
3  #   Rahul Nunna, 2017
4  #   Server socket class, for leader.
5  ##############################################################################
6
7  import socket
8
9
10 class serversocket:
11
12     def __init__(self, host, port=5454):
13         self._host = host
14         self._port = port
15         self._server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
16
17     def send(self, string):
18         self._server.sendto((string), (self._host, self._port))
19
```