UNIVERSITY OF HULL | FACULTY OF SCIENCE AND ENGINEERING

Turn virtual trajectory into reality by flying drones

Final Report
Submitted for the BSc in
Computer Science (Embedded Systems)

April 2019

by

Ieaun Roberts

Word count: 15 728

Abstract

Herein lies the background information and summarized findings of experiments, research and software development conducted to control the open source Crazyflie 2.0 drone (developed by Bitcraze) supported by a vision system (Vicon Tracker) which gives real time positional data that is processed through a regulated PID loop to produce torque values to alter the propeller speed of the drone. To allow users to interact with the device and specify a setpoint/trajectory for the drone to travel to, the following needs to be implemented, a client must be designed and within networking means must be used to intercept positional data from the Vicon server, this data must then be fed into a PID loop and transformed into a final value for propeller thrust. This project has been conducted to offer some insight into the basic development of regulatory systems and object tracking to give readers a foundation to build upon with their future work.

Original Project Description

"The goal of this project is to develop a PID controller for the Crazyflie flying robots supported by a global vision system. The work is to develop and optimise a PID controller for the drone and develop a graphic interface to interact with the drone so that the drone can fly and follow a given trajectory by the user. The user will play with the Crazyflie flying platforms and make use of the open source Bitcraze project as basics to build up a customised controller algorithm. The real time 3D coordinates of the robot can be obtained from the global vision system. It's an opportunity to work in the newly refurbished Robotics lab facilities with your sparkling talents. The project is interesting but challenging and a good level of maths and programming skills are desirable."

TABLE OF CONTENTS

CONTENTS

1. Introduction

Flight once thought of as an unattainable feat of nature, but now through great advancements in engineering, pioneering intellectuals such as **Sir George Cayley** (inventor of the first manned glider of which there is any record) have spearheaded us into our current day and age where items can be transported through interplanetary travel. By making use of numerous aeronautical innovations over the past few centuries their findings have allowed us to control mechanisms and their trajectory through use of virtual systems. One of these relatively recent innovations being flying quadcopter drones. (Crouch, 2018)

Within this document details the processes, findings and modifications made to the open source project developed, manufactured and sold by Bitcraze. The Bitcraze collective are the creators of a miniature quadcopter dubbed the Crazyflie, which upon completion was the smallest quadcopter in the world.

In this project the second iteration of the Crazyflie which was released in December 2014 shall be used to follow a given trajectory outline by the user via multiple set points. The CrazyFlie 2.0 has several improvements over the first including a socket for expansion boards and more powerful motors.

To add a navigational feature to the product, a graphical interface will be designed to allow users to seamlessly send flight data commands to direct the drone to a desired location. This client will make use of the Vicon vision system to track the CrazyFlie's movement and relay changes back to the firmware within the Flie to adjust its position. The data received from the user will enter a control loop mechanism called a PID controller within the client to send changes in propeller speed, yaw, pitch and roll according to data being returned by the Vicon vision system to reach a desired destination.

Importance

The occurrences on December 20th, 2018 at Gatwick airport which left over 140,000 passengers within roughly 1000 flights stranded for 36 hours of chaos has left its mark on the British aviation world. After multiple sightings of unmanned drones during the Christmas period on runways across the airport, it has needed a further 5-million-pound investment into emerging technologies to prevent future attacks.

Major international airports, including in the US, have been consulting Gatwick in the wake of the attack to find out how to prepare for drone attacks and improve their own responses. (Spero, 2019)

Advances in anti-drone technologies will inevitably breed invitation in the UAV field. This report on the complexities behind the software and hardware required will shed a much-needed light into the high-flying realm of autonomous flight using virtual trajectory.

1.1. What this project is about

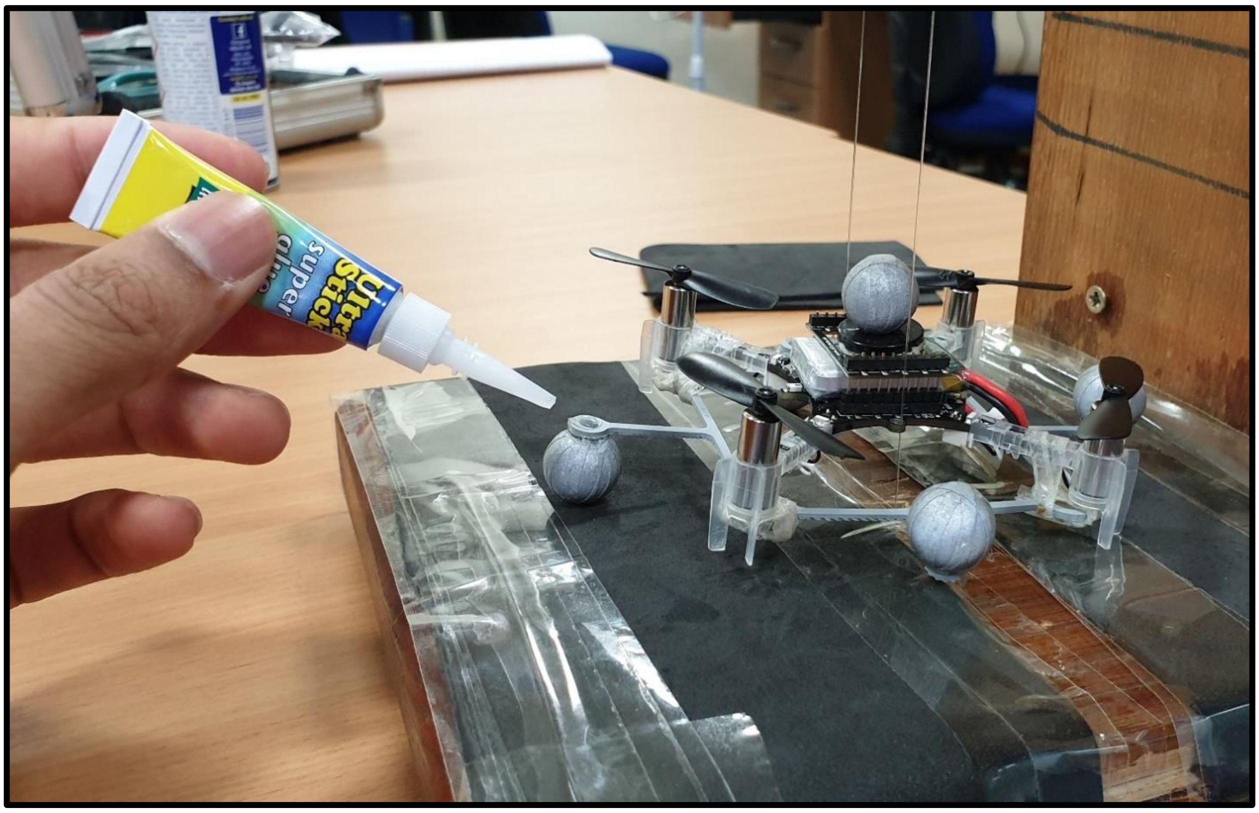
- Making use of the Crazyflie 2.0 to follow a given trajectory using multiple setpoints.
- Creation of client for logging and input of desired set points .
- Development of a PID controller to regulate yaw, pitch, roll and thrust variables within the Crazyflie firmware.
- Using the Vicon Tracking System to obtain the 3d co-ordinates and orientation of the drone, to send feedback to the PID controller so adjustments can be made to current values.

1.2. How this report is organized

The main content of this report is divided into 6 sub sections, each explaining different aspects of the project such as the background information that is necessary to fully understand the scope of the dissertation. They are:

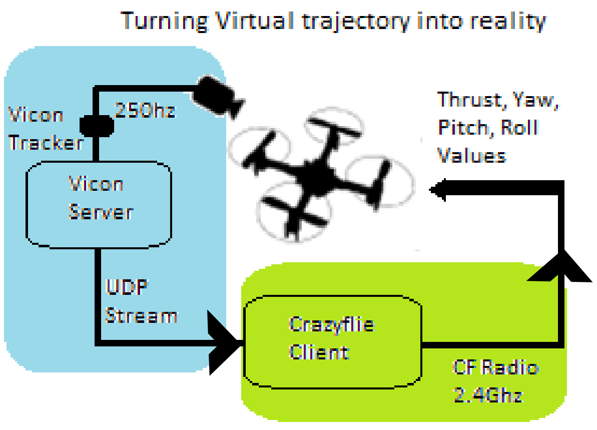
1. Introduction
2. Aims and objectives – defines the overall aim of the project
3. Background – general context of the project
4. Development – stages of development
5. Evaluation – evaluation of overall project achievements
6. Conclusion – finding of report summarized

*Note: Within this report the term "**Big 4**" refers to the values for thrust, pitch, yaw and roll*



[Figure 1, Crazyflie 2.0 drone equipped with custom frame and motion markers, secured in a wooden rig restricting flight to 1 dimension]

2. Aim and Objectives



The aim of this project is to customize the open source CrazyFlie PC client software system in a way that will allow users to transmit setpoint coordinates to the Crazyflie 2.0 drone. The Client will need to process these coordinates by making use of a PID controller which appropriately adjust thrust according to the Vicon vision systems feedback of the drone's current position to traverse a given path in a 1-dimensional space. After single dimensional traversal has been successfully achieved, the multipurpose PID controller structure will be used to regulate pitch, yaw and roll to stabilizes the drone in 3-dimensional flight.

Throughout the life cycle of this project 9 main objectives have been set out as:

Objective 1 – Conduct extensive research

- A considerable number of unknowns need to be researched before any visible deliverables can be presented:
  - Understanding PID controller operation
  - Quadcopter flight dynamics
  - The Python coding language
  - The C/C++ coding language
  - Virtualization software (Virtual Box or VMware)
  - Vicon Tracking System operations
  - Object tracking problem contexts
  - The Crazyflie python API
  - Qt design
  - Command line operations
  - ZMQ operations
  - The existing CrazyFlie Pc client code
  - TCP/UDP networking features
  - Structure of the CrazyFlie firmware
  - Effective averaging techniques on large amounts of frequently obtained data

Objective 2 – Design the Graphic User Interface

- A robust, dependable and ergonomic Graphical User Interface will need to be implemented as a bolt on to the original CrazyFlie client and assessed against usability / functionality criteria while being operated by a third party (Ethnography).

Objective 3 - Track the quadcopter using the Global Vision System

- By making use of the Vicon global vision system sensors, track the movements of the quadcopter in real time and send information back to the client so the data can be used in the PID controllers feedback arm to ensure stable flight by adjusting Big 4 values within the drone.
  - Build a frame that will support motion capture markers and attach to the Crazyflie to enable tracking.
  - Using UDP sockets, set up a network to receive the Vicon tracker stream
    - Parse the data received from the stream into usable data

Objective 4 – Use Injection commands to communicate with Crazyflie

- Use one of the discussed methods to communicate with the CrazyFlie similarly to how a gamepad/controller does but instead adjust this functionality by injecting set-points directly into the client and then sending adjustments in values to the drone.

Objective 5 - Develop and optimize (tune) a PID controller

- Develop and optimize a PID controller to regulate changes in thrust, yaw, pitch, and roll once given a command to travel towards a specific location.
- A user will enter their desired position in mm, this value will be approximated to a command in the client and sent to the drone.

Objective 6 – Optimize and refactor code for efficiency

- At this stage in the project many lines of code would have been written to perform 1 specific task independently of other tasks, the main goal of this objective is to unify all code and to optimize where possible.
    1. Optimize handling techniques used to intercept and parse Vicon data.
    2. Close the PID loop (using feedback data from the Vicon tracker).
    3. Send regulated thrust values to drone.
    4. Reconfigure drone markers if necessary.
    5. Test, test, test and then test some more.

Objective 7 – Travel to a desired setpoint in 1 dimension

- Using the PID controller regulating the torque of the propellers, travel to a desired set point

Objective 8 – Achieve flight using given trajectory in 1 dimension

- Making use of the drone, custom firmware, CrazyFlie PC client and vision system, command the drone to travel to a sequence of separate set points.

Objective 9 - Achieve flight using given trajectory in 3 dimensions

- Using stacked PID control loops for each of the *Big 4* variables, travel a given trajectory in a 3d space by making use of the rotational and positional data obtained via the Vicon UDP stream.

3. Background

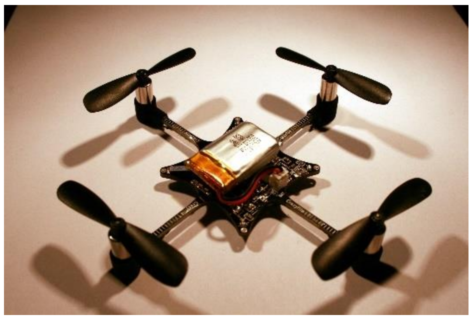
3.1. Drone vs Quadcopter

- Drone is a broad term used to describe aerial vehicles that do not require a pilot, also known as an Unmanned Aerial Vehicle (UAV).
- Quadcopter is a more specific term which refers to an aerial vehicle which takes flight by making use of 4 rotors.
- These 2 terms are used interchangeably throughout the project.

Key Hardware components to be used within this project

1. Crazyflie 1.0
    - Miniature quadcopter designed by Seeed with the intention in mind of giving users a versatile development platform that can be used to experiment, develop and explore many different areas of technology. (Bitcraze, 2019)
    - Contains components that need to be soldered into place.
    - Prone to damage due to impact from falling from high altitudes and collision.

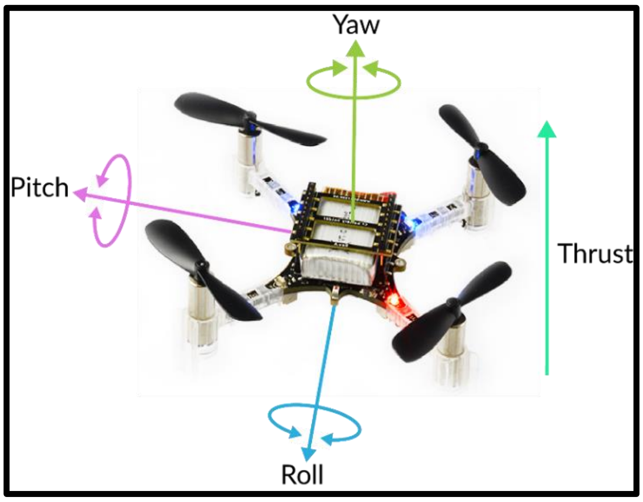
2. Crazyflie 2.0
    - Second iteration of the CrazyFlie quadcopter with many improvements.
    - Solderless
    - Used for research in many academic institutions for topics that fall under swarm research and trajectory control.
    - New safety fixtures around drone motors keeping vital mechanisms safe from impact damage. Less prone to constant repairs and can absorb more force before being significantly damaged.
    - Expansion boards: The CrazyFlie 2.0 has two rows of pin headers that allow for expansion decks to be attached on top of the quadcopter
        - 2.1) Flow deck v1/v2 - Uses sensors to measure the distance to the ground (2m/4m) to make precise omnidirectional trackable flight possible.
        - 2.2) Mocap deck - Allows tracking of movements in motion capture systems by making use of reflective markers.
        - More decks for specific tasks including Sd card adapters for extensive logging and a wireless charging pad. (Bitcraze, 2018)
3. Crazyflie 2.1
    - As of 18 February 2019, the 2.0 was discontinued and replaced with a new model that boasts improvements such as a radio amplifier.

Quadcopter	Crazyflie 2.0	Crazyflie 1.0
Appearance		
Weight	27 Grams	19 Grams
Flight time	7 minutes	7 minutes
Support for expansion boards	Multiple	Limited
Microcontroller	STM32F405, 168MHz, 192kb SRAM, 1Mb flash	STM32F103CB, 72 MHz, 20kb RAM,128kb flash
Measures taken to protect drone	Motor protectors, expansion boards are placed above battery keeping it locked in place	Minimal

[Figure 2, Comparison of different Crazyflie models, (Bitcraze, 2018) (Bitcraze, 2019)**]**

Concepts:
Thrust - A reaction force that can be described as a sudden accelerated push towards a single direction (In this project, positive thrust can always be assumed to be against the Y axis propelling it upwards).

Motion in a quadcopter is achievable in 3-Dimensions once thrust has propelled it off the ground.
Yaw - Rotation movement on the Y axis (Left/Right)
Pitch-Rotation Movement along the X axis in a specific direction (Forwards/Backwards)
Roll - Rotation Movement along the X axis in a specific direction (Up/Down)

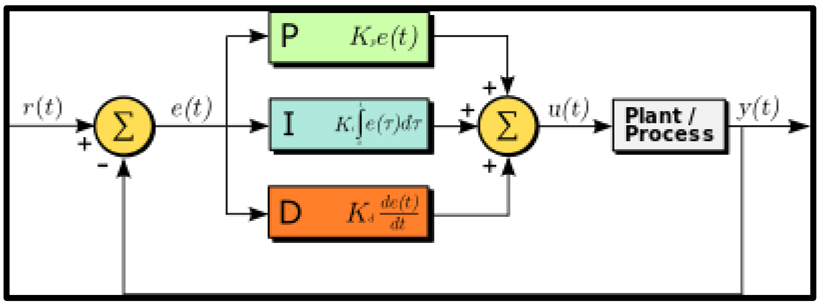


[Figure 3, Visual representation of the core concepts of quadcopter movement in the CrazyFlie 2.0 (Pri, 2018)**]**

3.1.1. PID Controller

3.1.1.1. Overview (The ideal PID controller)

A Proportional-Integral-Derivative controller (often referred to as three-thing control) can be described as a non-linear control loop, in which the difference between a desired **set point** (SP) and a measured position is calculated and adjusted according to proportional, integral, and derivative terms. Each letter represents an individual branch in the control process. Many different models exist where selected branches are either omitted or included (e.g., PI, PD…). {(Eurotherm,2018)} Controllers are not limited to aviation and are used in many fields across disciplines.



[**Figure 4, A PID controller in a feedback loop r(t)= Set Point, y(t)= Process value** (Urquizo, 2019)**]**

The purpose of the controller is to directly influence the plant system through **actuating signals** (input) to produce an acceptable **controlled variable** (Output).

Actuating signal
Plant
Controlled variable

[Figure 5, Plant system]

In operation a plant is fed a command variable dictating a goal (e.g. rise 50 meters). We refer to the **difference** between the **command variable** and the current value (received from the **feedback path)** as the **error term** (e.g. error term = 50 initially). The feedback path uses sensors to obtain measured state values with which it will determine how far the system is from the command variable after each successful loop and continuously correct the plant. So therefore, it is evident that the core purpose of our controller is to exponentially adjust the **actuating signal** according to the error term until, command variable - feedback path = 0 is true.

$$u(t) = K_{\mathrm{p}} e(t) + K_{\mathrm{i}} \int_0^t e(t')\,dt' + K_{\mathrm{d}} \frac{de(t)}{dt},$$

[Figure 6, Mathematical formula representing a PID control (Rabault, 2019)**]**

U= controller output signal
Kp = proportional gain
t = time
e = Error term
Ki = integral gain
Kd = derivative gain
d = derivative

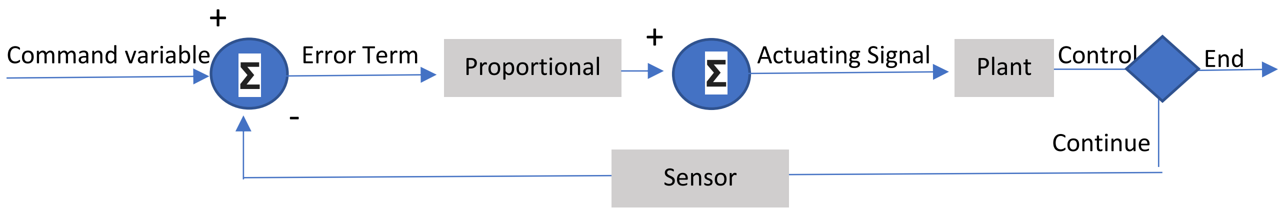
3.1.1.2. Proportional Response (Kp)

This branch will adjust the actuating signal proportionally to the size of the error signal. The proportional component depends only on the **difference between the set point and the process variable**. This difference is referred to as the **error term**. The proportional gain (Kp) determines the ratio of output response to the error signal. For instance, if the error term has a magnitude of 10, a proportional gain of 5 would produce a proportional response of 50 (National Instruments,2018)

There is a propeller speed at a certain amount of rotations per minute (RPM) where the lifting force is exactly equal to the weight of the drone of the propellers. This speed is needed for a drone to hover.

Problems related to proportional response:

- Steady State Error –As a system approaches the **Set point** the **error term** begins to shrink. Steady State Error is defined by the inability to reach an **error term** value of 0 through the multiplication of the proportional gain value and error term.
- E.g. a drone will hover once the propellers are spinning at 100rpm, we need to reach a height of 50 meters and we set the proportional gain value to 2.
  - Initially we begin with an error term that is equal to 50.
    - (Error term = **command variable - Error term**) or Error term = 50 – 0
  - Error term * proportional gain = propeller speed
    - In this case the drone will simply hover at ground level and not increase its altitude as 50 (command variable -error term) *2 (Proportional gain) = 100 RPM
- Similarly, if we were to change the proportional gain to a value of 5 or even 100 we would encounter the same Steady state error at 30 Meters and at 49 meters respectively since:
  - 20 (command variable – error term) * 5 = 100 RPM
  - 1 (command variable – error term) * 100 = 100 RPM
- Also, if an error value of 0 is reached the plant may send an actuating signal indicating the propellers to stop. Since 0 * any value = 0 and command variable = control variable.
- With only a proportional branch being altered to one of these values the 50-meter altitude goal cannot be reached, the error term could only ever be made smaller until it reaches steady state.
- This problem is unique to proportional gain and solved using **past error term** information in the Integral branch. (Douglas, 2018)



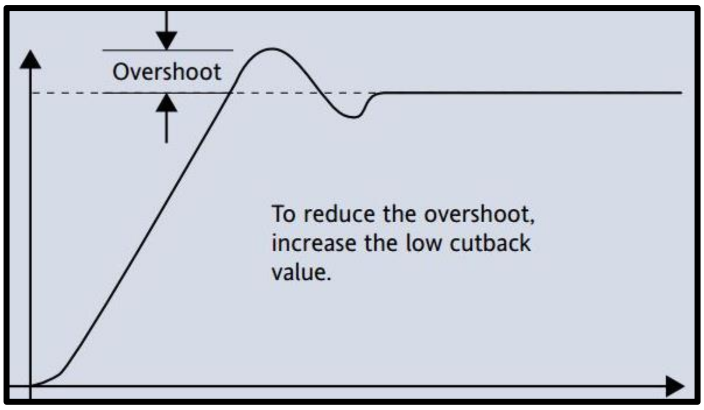
[Figure 7, proportional response, decision]

3.1.1.3. Integral Response (Ki)

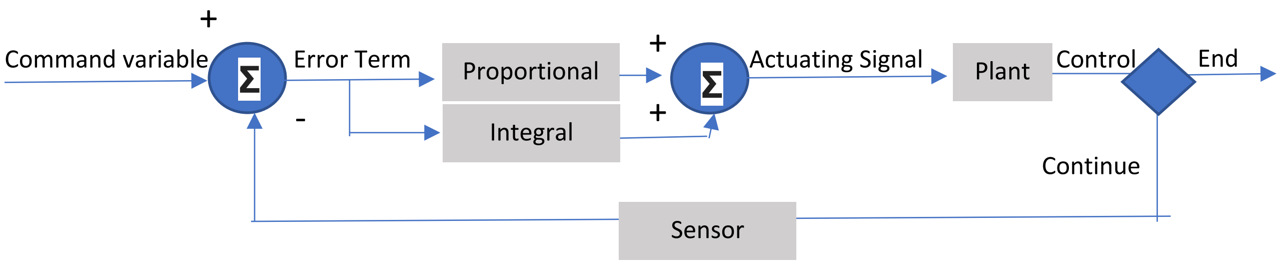
The integrator accumulates the error term over time resulting in even the smallest error term values increasing or decreasing the integral component gradually until the error term value is 0. Integral responses main goal is to drive steady state error to 0. While working with the proportional branch, if an error term value of 0 is reached the integral path ensures the drone continues to hover as the integral sum is outputting a steady value even if error = 0. (Graham C. Goodwin, 2018)

- All non-0 error term values get added to the integral running total, this increased value on this branch increases/ decreases the speed of the propellers.
- If there is error in the system, the integral sum will change.

On near arrival of the command variable value the proportional branch will be near 0 and have a very small contribution to the overall increase in RPM of the propellers. Depending on what values were used, the integral path may now be over the value required for the drone to hover. This will cause the drone to continue to rise and overshoot over the command variable value causing a negative error. When this value is added to the integrator it reduces the amount stored and slows the propellers down which causes the drone to once again drop below the goal. This can be prevented my making use of one final branch to predict when we will reach our goal (derivative response). (Douglas, 2018)



[(Eurotherm,2018), Figure 8, visualising the overshoot concept]



[Figure 9, Integral response]

Problems related to Integral response:

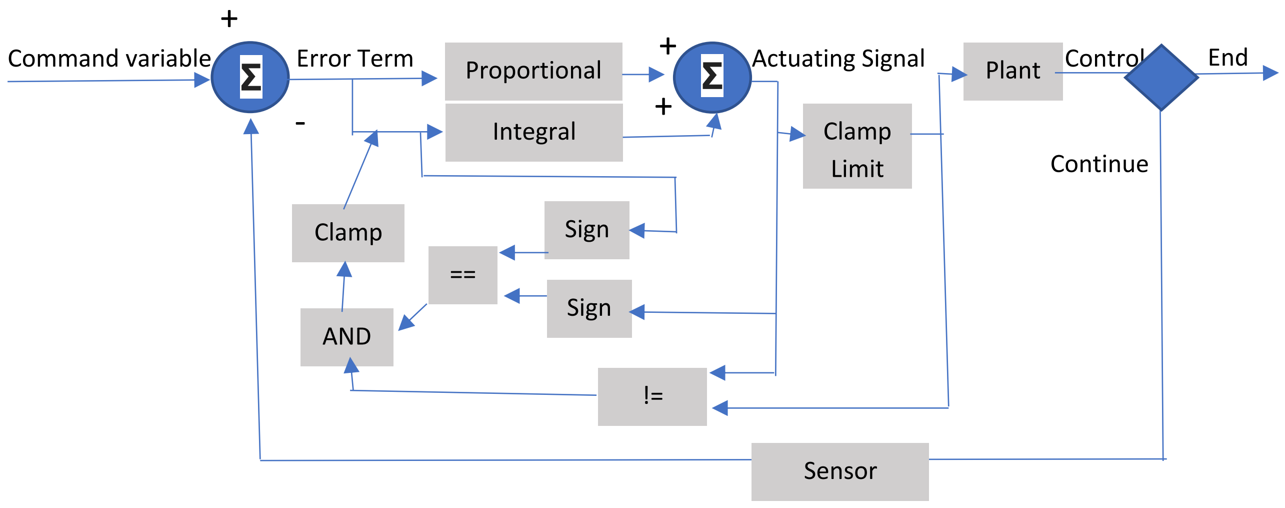
- **Integral Windup** – in a situation where a drone is unable to ascend, say it is held down by an undetermined force. The integral value will continue accumulating, increasing the actuating signal (higher RPM). This value will continue to rise until the force holding the drone down dissipates. Once the drone is free to ascended it has two major problems: (Douglas, 2018)

    1. The Rpm will now force the drone up in an erratic manner possibly causing a massive overshoot when working with a low command variable. (e.g. Rise 1 meter)
    2. The integral value will reach a point where even if it does subtract the negative error term from its accumulation decreasing the RPM, it shall continue to rise as its memory had accumulated all the error term values it should have been ascending by while it was being held

down. This results in the drone eventually descending after the negative error terms are subtracted from the integral.

   a. The drone's motor (actuator) will limit the amount of possible RPM saturation (the motor will not be able to exceed an amount of x RPM due to mechanical limits, any value requested over this limit cannot be achieved and is said to be saturating) but this is not enough to prevent this occurrence.

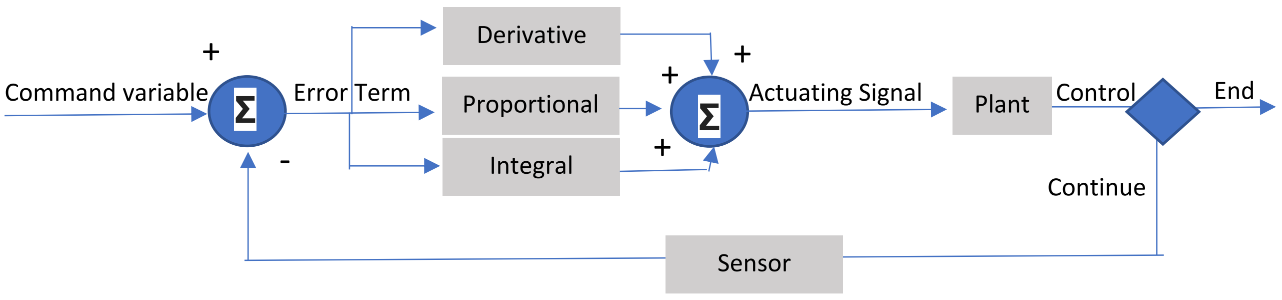
3. We can create a method to clamp the possible Integral saturation total value so that this problem can be minimized (E.g. set max RPM to 200)
   a. Past implementations have made use of 2 checks comparing the value before and after the clamping of saturation:
      i. Check if the values are equal.
         1. If equal then no clamping has taken place, if not equal then saturation is taking place
      ii. Compare the signs of the error term and actuating signal.
         1. If both positive then we know the integrator is still adding values increasing the total RPM rather than decreasing it, if both negative then we know the integral is attempting to make the value more negative, **IF error sign = control output sign**
   b. If the value is saturating and the integrator is causing the values to further increase/ decrease, we can then choose to completely clamp the Integrator path by setting its error term value to 0. Once the system is out of saturation or changes sign we can then restore the branch, so the values stored can begin to decrease. **This will instruct the drone to lower its motors rpm when it approaches it actuators limit.**



[Figure 10, Integral response with clamping]

3.1.1.4. Derivative Response (Kd)

The derivative measures rate of change in the error term to determine how fast we are reaching the goal and prematurely reduces the propeller speed. This prevents the problems outlined in the integral branch regarding the accumulation of saturated values. A system rising towards its goal has a negative rate of change as the error term is decreasing. This negative value is added to the output lowering the propellers speed preventing an overshoot. (Douglas, 2018)



[Figure 11, derivative response]

These concepts can be applied as follows:

- Actuating signal = rotations of propellers (Increase voltage to increase speed)
- Actuator = motors for propellers
- Controlled variable = current altitude/position
- Plant = CrazyFlie drone
- Error term = distance drone must still travel to satisfy instructions
- Command variable = trajectory co-ordinates given to system
- Feedback path/sensor = Vicon Tracker measures the position of the drone

3.1.2. Feedback

The feedback arm of our diagram represents the captured values of our system in relation to its position. A sensor will record how much the drone has moved in each axis. This recorded value in a real-world context will most likely contain noisy data.

Noise can be described as an unpredictable disturbance on a signal that is unavoidable. Noise can be attributed to anyone of either environmental, implementational or simply a defect in the sensor hardware. To lessen the impact on what values we are receiving from the feedback loop a filter must be created to ignore values over a certain frequency or lessen the effect that these noisy values have on our system.

True state → sensor → Measured state

Measured != True state

3.1.3. Additional information:

Kalman filter – allows a user to extract information from a situation that cannot be precisely measured due to noise using estimation. It is an optimal estimation algorithm that predicts parameters of interest in the presence of noise in measurements such as speed or direction. (Grewal, 2001)

Past implementations of PID systems

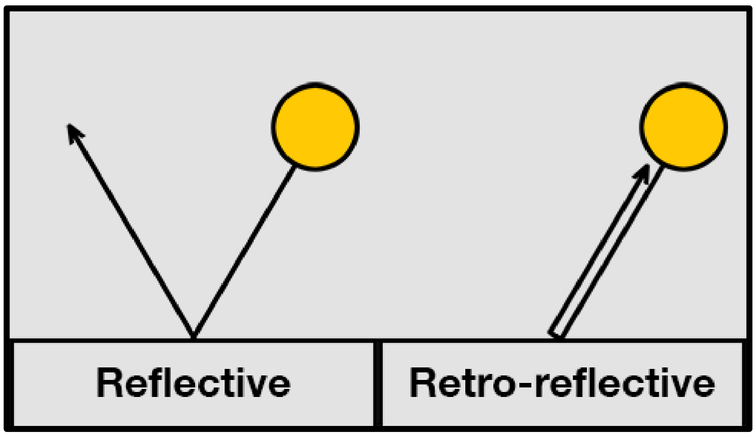
- Most instances of what is known as “Cruise control” within the automobile industry use PID control to regulate the acceleration of the vehicle without overshooting.
- In heating control units throughout many homes across the world PID controllers are used to heat water to a desired constant temperature.

3.2. Hardware Systems

3.2.1. Mo-Cap Markers

To aid with the tracking of the drone mo-cap (motion capture) markers were obtained. The mo-cap markers used in this project are light weight, spheres coated with strips of a material known as "retro-reflective" material.

Retroreflective material works differently from normal reflective or shiny materials. Normally light bounces off a surface when it's reflected, away from the light source. But retro-reflective materials reflect light back in the direction it came. The most common way it does this is by using tiny glass beads embedded within fabric. (Maliszewski, 2015)



[Figure 12, Retro-reflective vs Reflective]

3.2.2. Vicon Global Vision System (Vicon Tracker)

The Vicon Tracker system is an accurate camera system that will be used to track the CrazyFlie's position mid-flight and send coordinate information back to the PID controller. Vicon are the premier solution for UAV and robotics studies. Their systems are described as highly accurate and easy to use. (Vicon, 2018) In this project 4 Bonita B10 cameras will be used as sensors to track the custom made mocap deck attached to the drone and send data to the Vicon host pc which acts as a server sending data outwards to clients.

Device	Frames Per Second (FPS)	Resolution	Image formation type
Bonita B10 (Vicon x4)	250 FPS	1 Megapixel	Infrared
Kinect 360/ X-Bone	Both 30 FPS	640x480, 3.6 Megapixels /640x480, 12 Megapixels	Infrared
Average laptop web cam	30 FPS	3 Megapixels	Visible light

[Figure 13, Vicon vs Kinect vs Webcam]

3.3. Software Systems

3.3.1. PyCharm

This is the Python IDE that will be used to further develop the CrazyFlie client. The development environment has built in management capabilities for version control and autocomplete features which reduce the amount of time coding simple statements. (JetBrains, 2019)

Python version 3.6 was used in client development and is available for download at : https://www.python.org/ (foundation, 2019)

3.3.2. Qt Designer

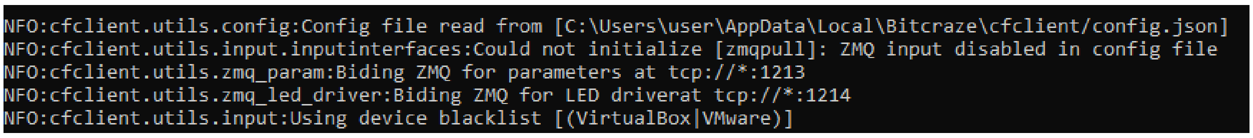
Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets. You can compose and customize your windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner and test them using different styles and resolutions. (QT, 2019)

3.3.3. ZMQ

ZMQ is a high-performance messaging library which can be used to interconnect "every modern language" across multiple platforms. It operates by carrying messages that make use of the TCP/IP protocols. It is backed by a large active community and is used in many open source projects across various fields. (Wikidot, 2019)

In the project context, ZMQ is used as one of the means of communicating between the client and drone by transporting data objects using TCP/IP. It is used to send big 4 values from the client to the CF 2.0.

To control the drone via ZMQ commands the configuration file inside the clients file structure must be altered to read for said commands at start up. There are 2 configuration files, "//home/Bitcraze/.config/cfclient/" and "/home/bitcraze/Desktop/projects/crazyflie-clients-python/src/cfclient/configs/" within the Bitcraze VM regarding ZMQ. Originally, only the later was altered and was met by inactivity from the client. Altering the first file fixed this, read the run time console within PyCharm to examine whether your client is ZMQ enabled at start-up. (arnaud, 2019)


```
NFO:cfclient.utils.config:Config file read from [C:\Users\user\AppData\Local\Bitcraze\cfclient/config.json]
NFO:cfclient.utils.input.inputinterfaces:Could not initialize [zmqpull]: ZMQ input disabled in config file
NFO:cfclient.utils.zmq_param:Biding ZMQ for parameters at tcp://*:1213
NFO:cfclient.utils.zmq_led_driver:Biding ZMQ for LED driverat tcp://*:1214
NFO:cfclient.utils.input:Using device blacklist [(VirtualBox|VMware)]
```
[Figure 14, Represents the console during start-up of a client without ZMQ enabled]

3.3.4. VRPN

The Virtual-Reality Peripheral Network (VRPN) is a library that provides an interface between 3D immersive applications and tracking systems used for Virtools. The Vicon Tracker system has a built-in VRPN server that will stream data natively into these applications or will allow for the development of simple interfaces using VRPN. (Limited, 2018)

3.3.5. VirtualBox

VirtualBox is the virtualization software Bitcraze recommends using to run their virtual machine for the development of code for their device. Virtual Box is a free open source hosted hypervisor (creates and runs virtual machines). (VirtualBox, 2019)

Within this project the virtual machine used is running Linux Ubuntu 18.04 which is provided by Bitcraze along with multiple programs needed for drone development. These programs include Eclipse for C++/C development of firmware and Qt designer for UI design but strangely enough does not include a python IDE such as PyCharm to alter their clients code (This could be an attempt to decrease the overall size of the virtual machine or licencing issues), this was installed separately.

3.3.6. CrazyRadio PA

The CrazyRadio PA is a 2.4 GHz radio transmitter USB dongle with a power amplifier. Currently it's only used to communicate with the Crazyflie, but it can be adopted to work with several applications that are based on the low-cost 2.4 GHz chips from Nordic Semiconductor. Although this device is needed for desktop control of the CrazyFlie, a mobile app version of the CrazyFlie Client exists that makes use of an android or IOS devices Bluetooth low energy signal to transmit data to and from the quadcopter. (Bitcraze, 2019)

3.3.7. Crazyflie python library (CFLIB)

CFLIB is an API written in python used to communicate between the quadcopter and client. Within its file structure are many .Py files containing useful methods. Its functionality includes flashing of firmware, functions to alter Big 4 values, functions to send absolute positional data obtained through vision devices such as the Vicon Tracking system to the drone's firmware and many others. (Bitcraze, 2018)

3.3.8. Crazyflie PC Client

The client created by Bitcraze for the quadcopter which gives users the ability to control the Crazyflie using peripheral devices such as a PS4 controller or even your mouse (not recommended). The backbone of this client was used and further developed throughout this project to add functionality not already present within the current version of the client (That functionality being autonomous flight and processing/display of external positional data).
Libraries used in development:

- NumPy (Stéfan van der Walt, 2011)
- Matplotlib (Hunter., 2007)
- Crazyflie API (Bitcraze, 2018)

3.4. Technical Background

3.4.1. Language comparisons

For the development of the firmware for the Crazyflie quadcopter, research was conducted into various methods and features of C, C++ and C# to find the optimal language that could portably assemble code, process a continuous amount of feedback data quickly and is optimal for use in memory constrained devices such as the CrazyFlie 2.0 which only has 192kb of SRAM and 1Mb flash. This initially ruled out C# as it has a high-level of abstraction and isn't optimal for embedded devices, leaving only C and C++.

C	C++	C#
Lowest -Level of abstraction	High-Level but considered low when compared against languages like Java and C#	High-Level abstraction
Can code for any platform	Can code for any platform	Targeted towards windows OS
Great for embedded devices	Good for server-side applications and gaming	Good for simple web and desktop applications

[Figure 15, Comparison of languages for firmware Development (Wodehouse, 2018)**]**

As the description of the dissertation topic clearly states unambiguously, "*The user will play with the Crazyflie flying platforms and make use of the open source Bitcraze project as basics to build up a customised controller algorithm.".* The language that will be used to optimize the PID controller and make use of all other features of the CrazyFlie's hardware will be C. C is great for embedded devices as it is a low-level language and is an efficient language when working with other drivers. The code that is supplied by Bitcraze for its quadcopter's firmware is written in the C language. The selected C IDE will be eclipse.

Similarly, the CrazyFlie PC Client supplied by Bitcraze shall be customized according to what is specifically needed by this system rather than creating a new client and losing all the functionality already in place such as peripheral controller support and CrazyRadio PA implementation already in place. The CrazyFlie PC client is written in Python which as a language compared to C/C++ has the following advantages. C/C++ is slow to write, error-prone, and frequently unreadable, Python is known for its writability, error reduction, and readability. (Radcliffe, 2018)

Prerequisites that were needed before any substantial software development could be done:

- Ability to code in C (Learnt in 3rd year)
- Ability to code in Python
- Experience with command line, Linux based systems and virtual systems

3.4.2. Box Muller Method

This method produces 1 random number in a uniform distribution. It returns 1 of 2 random numbers that are each normally distributed using the provided mean and standard deviation. Increasing these values introduces more noise into the simulation system created. (Goodman, 2005)

3.4.3. Sliding window technique

In the sliding window method, a window of specified length, Len, moves over the data, sample by sample, and the statistic is computed over the data in the window. The output for each input sample is the statistic over the window of the current sample and the Len - 1 previous samples. In the first-time step, to compute the first Len - 1 outputs when the window does not have enough data yet, the algorithm fills the window with zeros. In the subsequent time steps, to fill the window, the algorithm uses samples from the previous data frame. The moving statistic algorithms have a state and remember the previous data. (Bodenham, 2019)

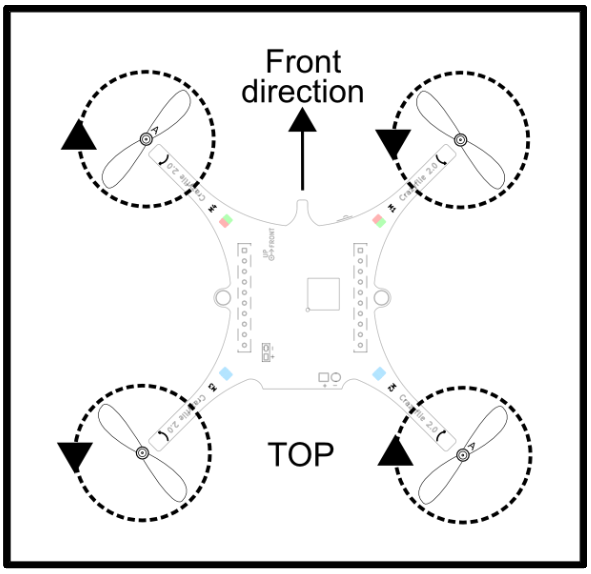
3.5. Additional Background

3.5.1. Bitcraze wiki/ Bitcraze Forum

These pages contain most and very near to all the information available on the CrazyFlie drones, their peripherals and implementation of certain features. Use of these resources is paramount to successful completion of any project making use of their drone.

3.5.2. Crazyflie Flight Dynamics

The Crazyflie has 4 equally spaced rotors arranged at the corners of its rigid square body which are each controlled independently. Each produces a torque (propeller rotation speed) and thrust (reaction force accelerating in one direction) around it's a centre of rotation. Propellers must spin in opposite direction pairs rotating with both rotating either clockwise or counter clockwise as evident in the figure bellow. Yaw, pitch and roll are all altered by adjusting the torque of individual rotors to create an imbalance. (Gibiansky, 2019)



[Figure 16, Propeller rotation directions and orientational information of the CrazyFlie drone (Bitcraze, 2018)**]**

4. Development

Development was an arduous process with many possible methods that could have been used in each segment of the project examined/implemented.

Methods for controlling the CrazyFlie:

- Altering parameters through client
- Sending data via ZMQ
- Sending commands using the Crazyflie API
- Sending external positional data and 3D positional commands to the drone's firmware using the Crazyflie API

Methods for accessing the Vicon Data Stream

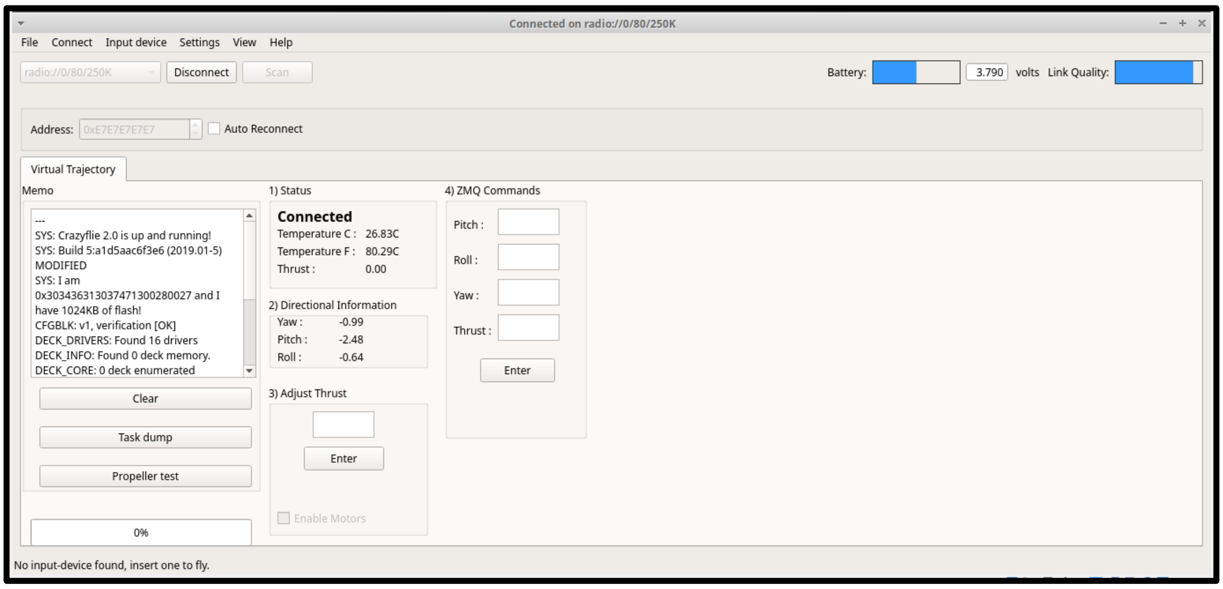
- VRPN
- UDP
- Vicon Tracker SDK

Client VS Firmware PID implementation

- Client based
- Firmware base

4.1. Client Design

Along the development of the project the client's appearance and functionality was changed depending on what methods were currently in place and what input was needed from the user evident by the figures below showing progress from the initial use of ZMQ for setpoint injection to the CrazyFlie APIs setpoint commands.



Connected on radio://0/80/250K

radio://0/80/250K

Battery: 3.790 volts Link Quality:

Address: 0xE7E7E7E7E7 Auto Reconnect

Virtual Trajectory

Memo

```
---
SYS: Crazyflie 2.0 is up and running!
SYS: Build 5:a1d5aac6f3e6 (2019.01-5)
MODIFIED
SYS: I am
0x3034363130374713002800027 and I
have 1024KB of flash!
CFGBLK: v1, verification [OK]
DECK_DRIVERS: Found 16 drivers
DECK_INFO: Found 0 deck memory.
DECK_CORE: 0 deck enumerated
```

1) Status

Connected

Temperature C :	26.83C
Temperature F :	80.29C
Thrust :	0.00

2) Directional Information

Yaw :	-0.99
Pitch :	-2.48
Roll :	-0.64

3) Adjust Thrust

Enable Motors

4) ZMQ Commands

Pitch :

Roll :

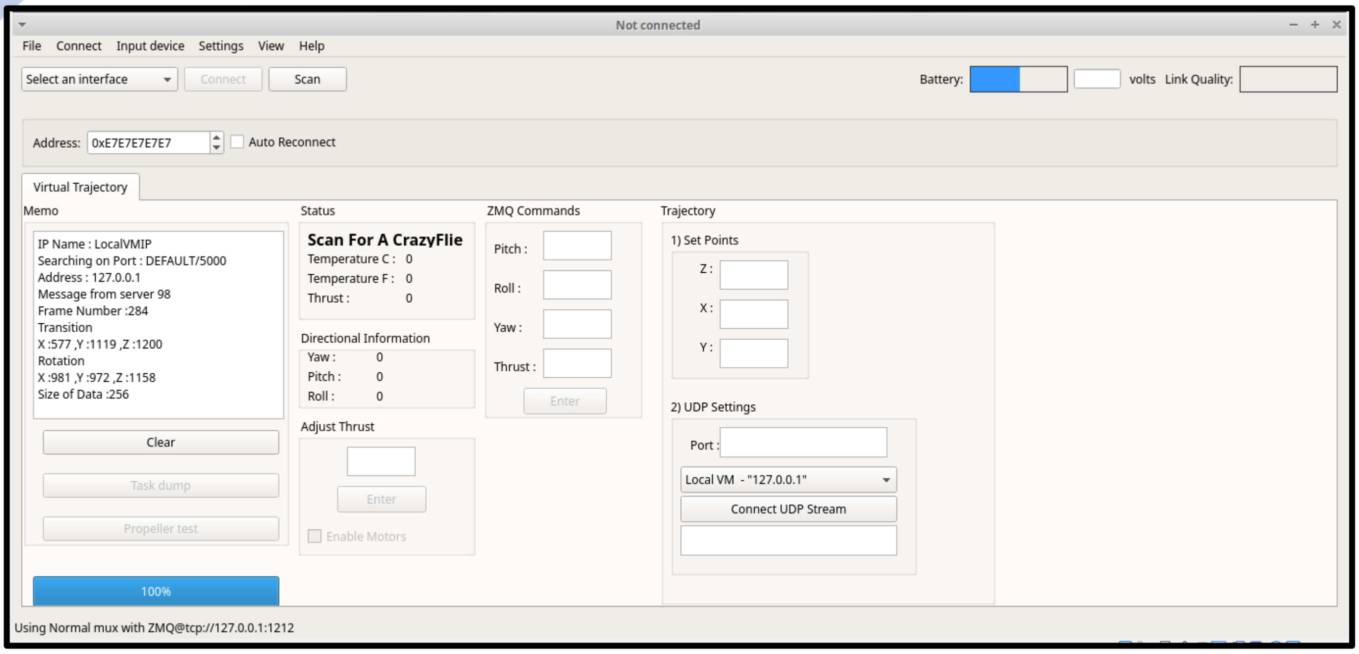
Yaw :

Thrust :

0%

No input-device found, insert one to fly.

[Figure 17, Client development progress 10/12/2018]



Address: 0xE7E7E7E7E7 Auto Reconnect

IP Name : LocalVMIP
Searching on Port : DEFAULT/5000
Address : 127.0.0.1
Message from server 98
Frame Number :284
Transition
X :577 ,Y :1119 ,Z :1200
Rotation
X :981 ,Y :972 ,Z :1158
Size of Data :256

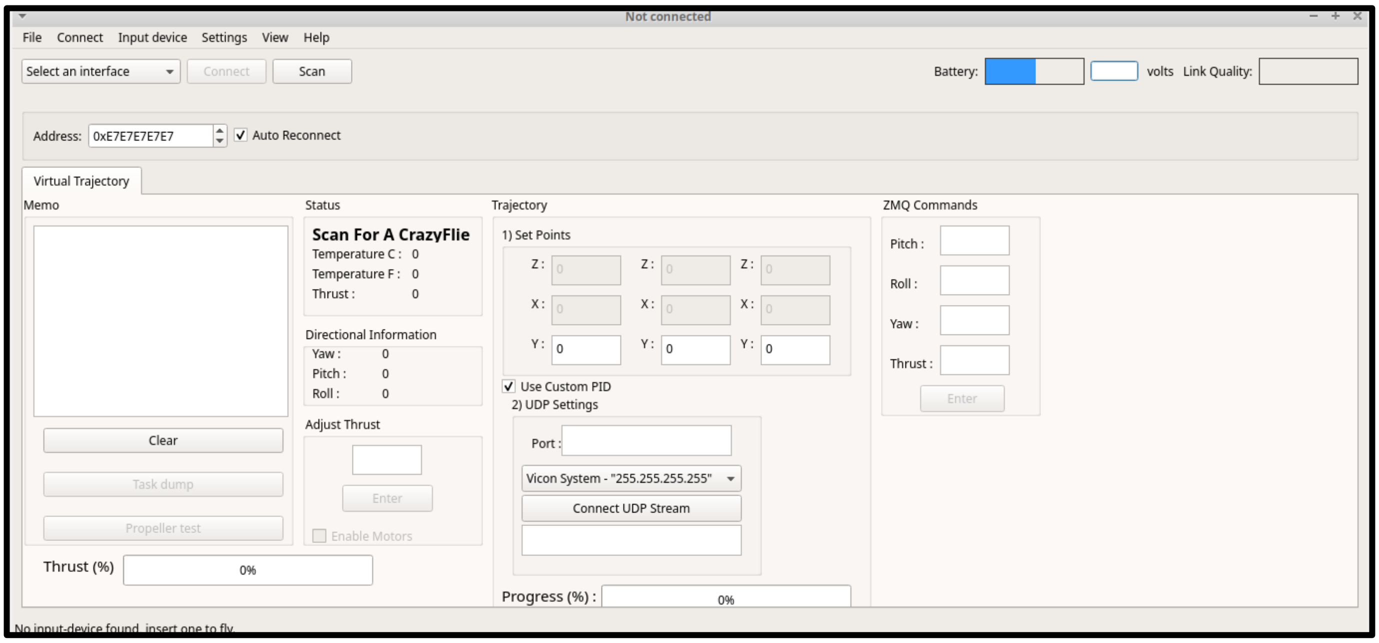
Scan For A CrazyFlie
Temperature C : 0
Temperature F : 0
Thrust : 0

Yaw : 0
Pitch : 0
Roll : 0

Local VM - "127.0.0.1"

Using Normal mux with ZMQ@tcp://127.0.0.1:1212

[Figure 18, Client development 10/03/2019]



Address: 0xE7E7E7E7E7 ✓ Auto Reconnect

Scan For A CrazyFlie
Temperature C : 0
Temperature F : 0
Thrust : 0

Yaw : 0
Pitch : 0
Roll : 0

Z : 0 Z : 0 Z : 0
X : 0 X : 0 X : 0
Y : 0 Y : 0 Y : 0

✓ Use Custom PID

Vicon System - "255.255.255.255"

Thrust (%) 0%

Progress (%) : 0%

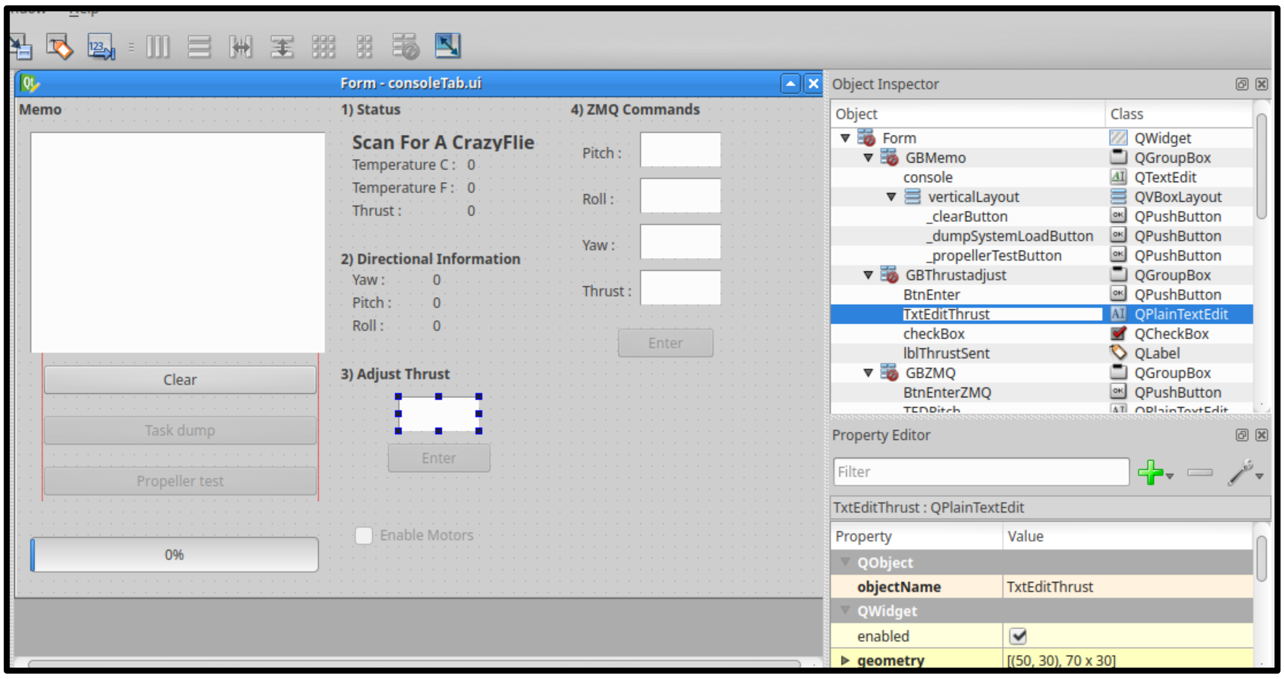
No input-device found, insert one to fly.

[Figure 19, Final Client design 20/04/2019]

In the final version of the client, users have the ability to enter a one-dimensional trajectory (on the Y-axis) consisting of 3 setpoints for the drone to follow. The command is sent once the user clicks "Connect UDP stream".

- Unticking the "Custom PID" will cause the client to attempt to use the experimental method 4 type control, discussed below.

Using QT Designer, a custom tab has been created inside of the existing client to perform logging operations and allow input to be sent to the CrazyFlie drone using the Crazyflie python API.



Temperature C : 0
Temperature F : 0
Thrust : 0

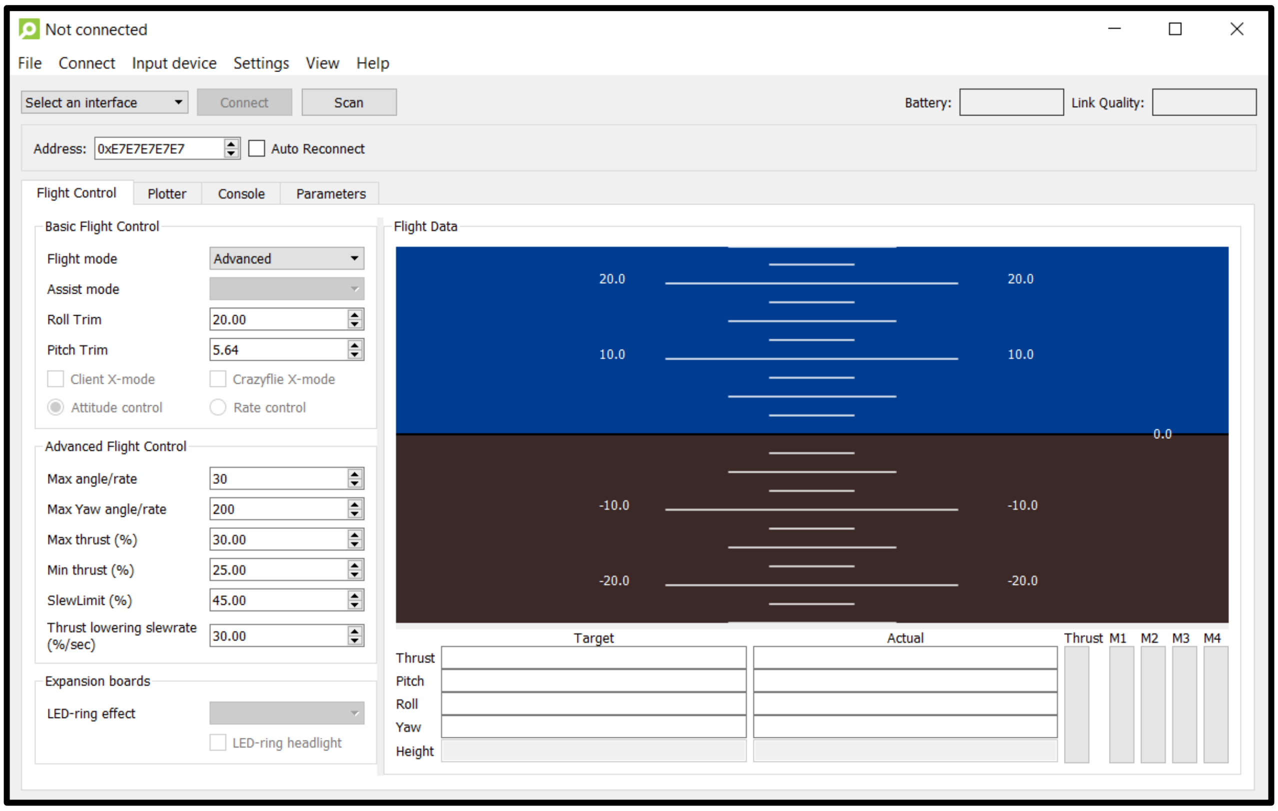
Yaw : 0
Pitch : 0
Roll : 0

Object	Class
Form	QWidget
GBMemo	QGroupBox
console	QTextEdit
verticalLayout	QVBoxLayout
_clearButton	QPushButton
_dumpSystemLoadButton	QPushButton
_propellerTestButton	QPushButton
GBThrustadjust	QGroupBox
BtnEnter	QPushButton
TxtEditThrust	QPlainTextEdit
checkBox	QCheckBox
lblThrustSent	QLabel
GBZMQ	QGroupBox
BtnEnterZMQ	QPushButton

TxtEditThrust : QPlainTextEdit

Property	Value
QObject	
objectName	TxtEditThrust
QWidget	
enabled	✓
geometry	[(50, 30), 70 x 30]

[Figure 20, Qt development of the Trajectory tab]



Select an interface

Address: 0xE7E7E7E7E7 Auto Reconnect

Basic Flight Control

Flight mode	Advanced
Assist mode	
Roll Trim	20.00
Pitch Trim	5.64
Client X-mode	Crazyflie X-mode
Attitude control	Rate control

Advanced Flight Control

Max angle/rate	30
Max Yaw angle/rate	200
Max thrust (%)	30.00
Min thrust (%)	25.00
SlewLimit (%)	45.00
Thrust lowering slewrate (%/sec)	30.00

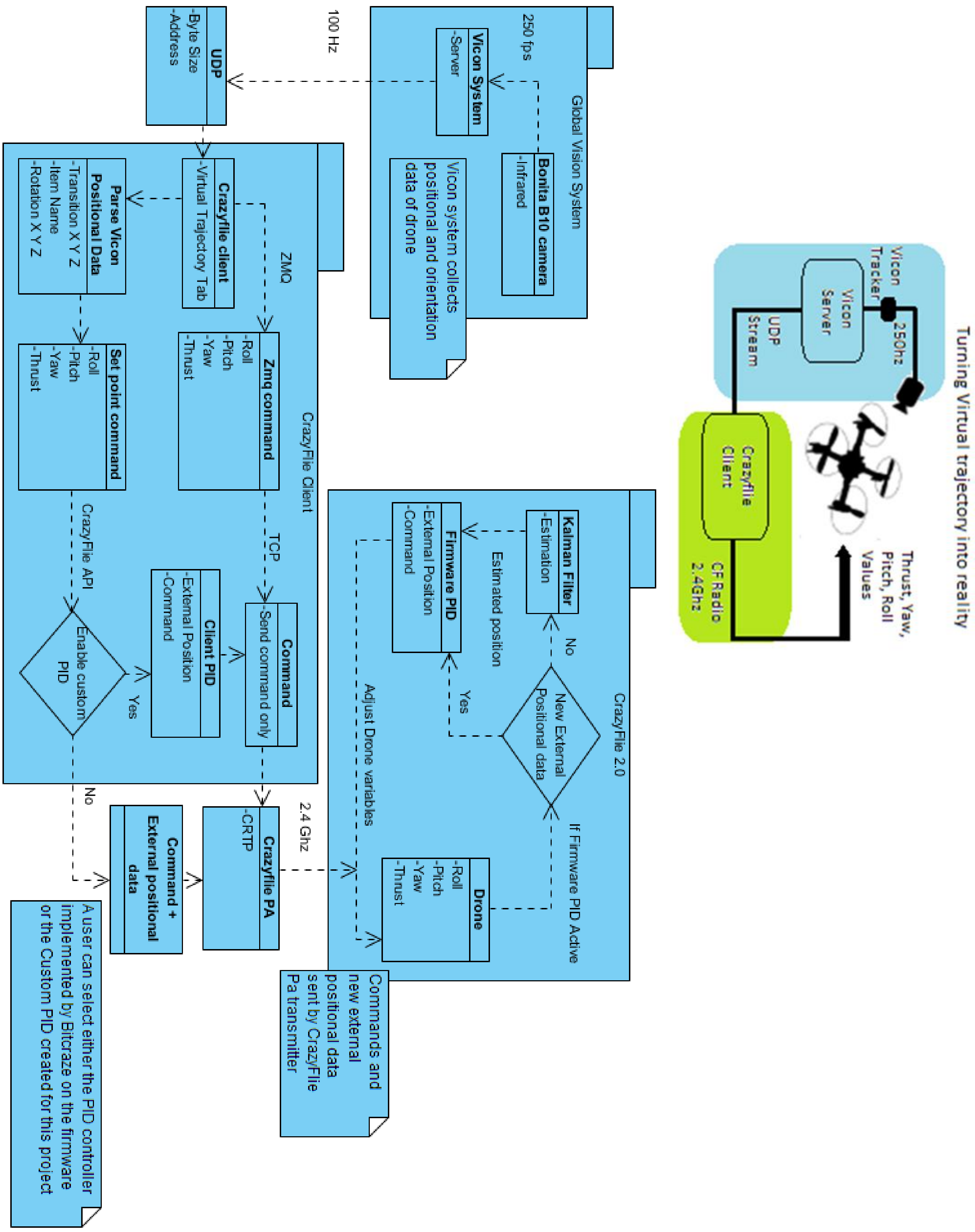
Expansion boards

LED-ring effect

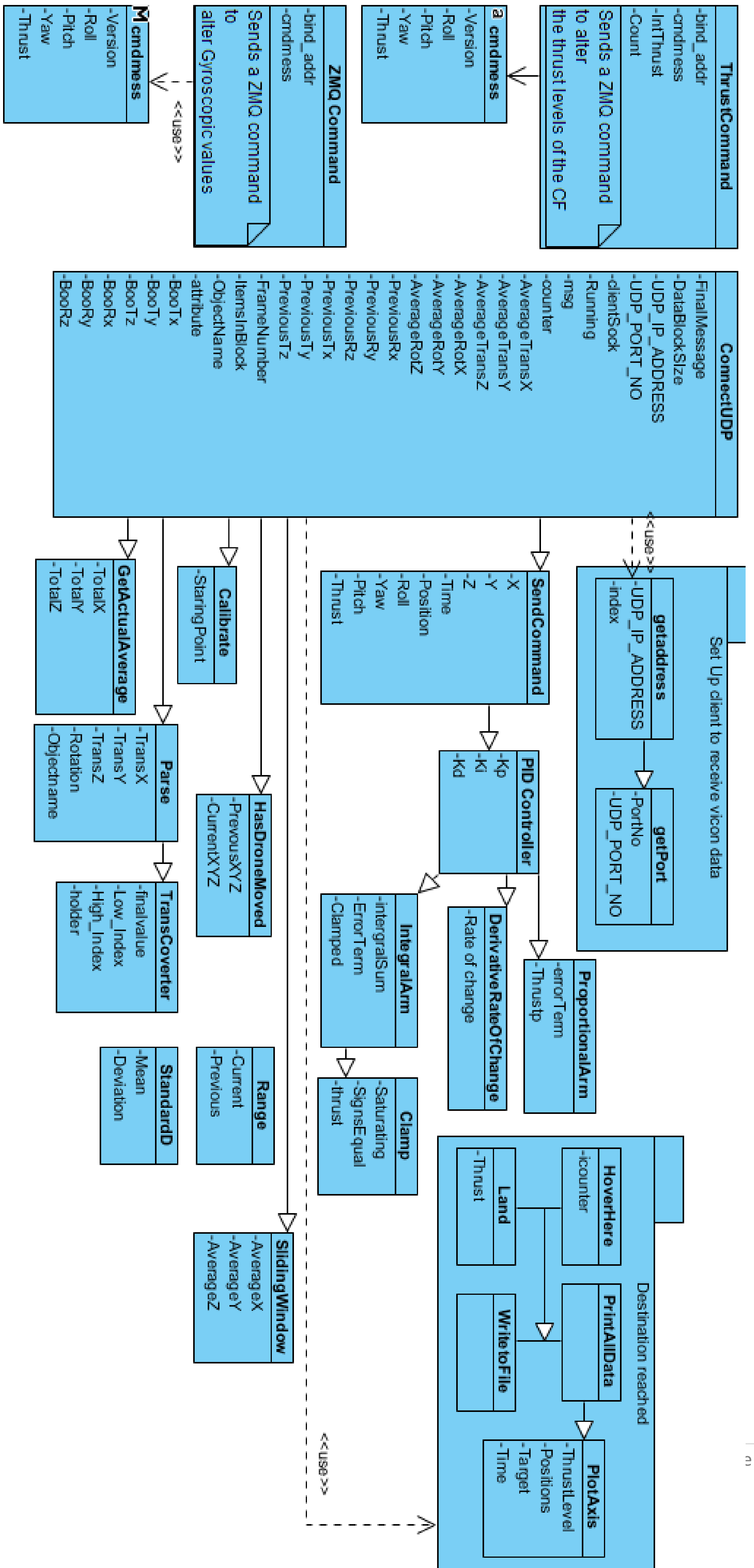
LED-ring headlight

[Figure 21,The Bitcraze CrazyFlie PC Client in operation]

SYSTEM DESIGN



Class Design



Function name	Basic Description
ThrustCommand	Sends a Zmq command to alter thrust levels
ZMQCommand	Sends a Zmq command to alter thrust, yaw, pitch and roll
ConnectUDP	Encapsulates all functions needed to control the CrazyFlie to travel to a specific position while making use of Vicon positional data.
getAddress	Read input from user regarding address, if none set to default.
getPort	Read input from user regarding port, if not set to default.
SendCommand	Start PID Controller after obtaining all needed data.
PIDCOntroller	Call all induvial PID arms.
ProportinalArm	Multiply the error term by Kp
IntergralArm	Add the error term value to the integral sum and multiply by Ki
DerivativeRateOfChange	Calculate the rate of change and multiply by Kd
Clamp	Determines if the integral arm must be clamped
Calibrate	Calculate the current position of the CrazyFlie before any commands are sent to get the starting point
HasDroneMoved	Using previous positional data, calculate if the drone's position has changed enough to confirm that it has moved and is not just noise.
Parse	Parse Vicon byte data into various variables such as transx for translation data
SlidingWindow	Add a new value to an array, subtract the first value of the array, calculate the sum of that array.
GetActualAverage	Calculate the average of a set of data.
TransConverter	Convert individual byte data into integer and sum all values for a specific variable e.g. transx
HoverHere	Set thrust to the hover value and send repeatedly for a specified amount of time
PrintAllData	Print all data within client and IDE
WritetoFIle	Write log data to file
Land	Send decrementing values of thrust to the drone to make it land
PLotAxis	Plot obtained data
Range	Will return a bool determining if a current value is +- in range of a previous value
StandardD	Calculates the standard deviation of a set of data

[Figure 22, basic descriptions of each function, (All code contains in depth comments on the majority of the lines written, for more specific information on a function view the codes inline comments)]

4.2. Controlling the CrazyFlie

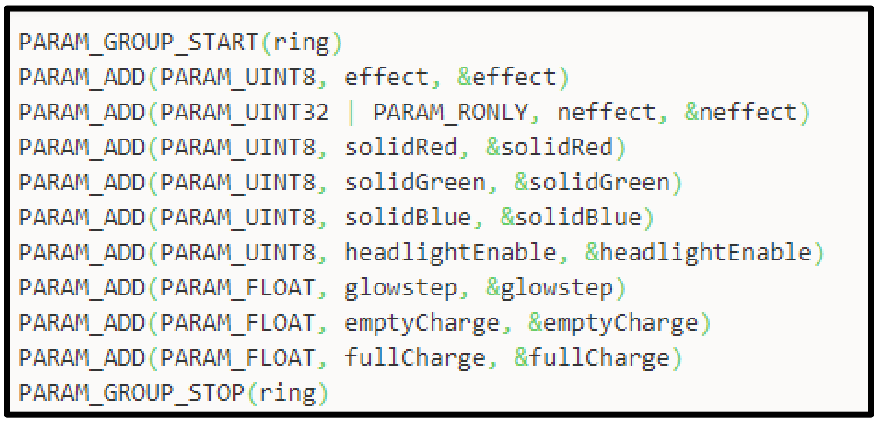
To have control over the ***Big 4 variables*** (thrust, pitch, yaw, rotation) is to have control over the drone. With the CrazyFlie being open source and having multiple homebrew version across many languages available, it results in many possible ways of controlling these variables. Here we discuss 4 of the methods found on the Bitcraze wiki / the dark depths of the Bitcraze forum and evaluate each.

Note: all the methods mentioned make use of the Bitcraze Crazyflie PC client which contains all the dependencies and libraries needed to establish communication between the user and drone. This includes the library named Cflib which handles events such as connection handshakes and exchanging data. The Python files named "commander.py" and "extpos.py" within the library handle 2 of the events discussed below and are worth examining further for use in future development

When setting the thrust levels be aware that a thrust level under 15000 will result in partial motor control, (only 3/4 or 2/4 of the motors will rotate and the drone will not create enough lifting force to fly).

Rule of thumb: use the CrazyFlie Mobile Client on IOS and Android to test any issues if you feel there is a problem with your drone, if problems do not persist when using the app then the problem is in your implementation or the transmitter used to send the signals to the drone attached to your Pc.

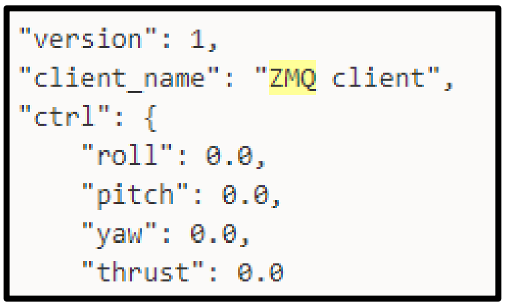
4.2.1. Methods used to Control with the CrazyFlie

1. **Altering parameters through the client** – the client contains within it means to monitor what is referred to as a Toc (Table of contents), this Toc is a group of parameters/logging framework within the Crazyflie firmware that can be made use of to alter and monitor specific variables, most of these variables have access restrictions preventing the drone from failing mid-flight. When the client connects it downloads the Toc for both logging and parameters.
    a. **Disadvantage –**the Bitcraze wiki states "There's no thread protection on reading/writing. Since the architecture is 32bit and the largest parameter you can have is 32bit it's safe to write one variable. But if you write a group of variables that should be used together (like PID parameters) you might end up in trouble" (Bitcraze, 2019)
    b. 

```
PARAM_GROUP_START(ring)
PARAM_ADD(PARAM_UINT8, effect, &effect)
PARAM_ADD(PARAM_UINT32 | PARAM_RONLY, neffect, &neffect)
PARAM_ADD(PARAM_UINT8, solidRed, &solidRed)
PARAM_ADD(PARAM_UINT8, solidGreen, &solidGreen)
PARAM_ADD(PARAM_UINT8, solidBlue, &solidBlue)
PARAM_ADD(PARAM_UINT8, headlightEnable, &headlightEnable)
PARAM_ADD(PARAM_FLOAT, glowstep, &glowstep)
PARAM_ADD(PARAM_FLOAT, emptyCharge, &emptyCharge)
PARAM_ADD(PARAM_FLOAT, fullCharge, &fullCharge)
PARAM_GROUP_STOP(ring)
```

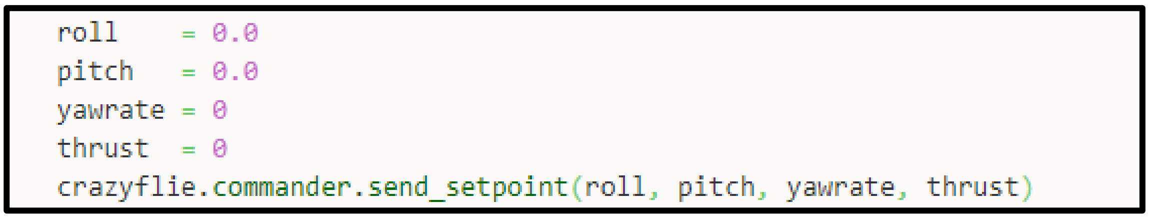
[Figure 23, Parameter group code written in C within the CrazyFlie firmware]

2. **Sending data via ZMQ** - ZMQ can be used to send data to the drone via TCP. One of the benefits of ZMQ is in its simplicity, users need only specify the variable they want to alter along with its new value.
    a. Within the firmware of the drone is a safety lock which will not let the user adjust the thrust value unless a Thrust value = 0 command is sent initially to unlock the ability to alter thrust.
    b. **Disadvantage** – one of the major problems linked to ZMQ is occasional loss of packets (The nature of ZMQ) resulting in an unresponsive drone and an **excessive loss of packets resulting in a disconnect from the drone**.
        i. In operation the use of TCP ensures that any lost packets are added to the front of the backlog of commands and resent. Problems arise when multiple commands are lost, as with our PID controller thrust torque levels are constantly changing to adjust position very frequently. Delays caused by resent packets eventually cause accumulations in the integral sum of the Ki arm and adversely affect the performance of the controller.
        ii. Inevitably after a large majority of the work had been based around ZMQ it was reluctantly ousted due to its fatal disconnect triggered by excessive packet loss.
    c. **[Figure 24, ZMQ command structure]**



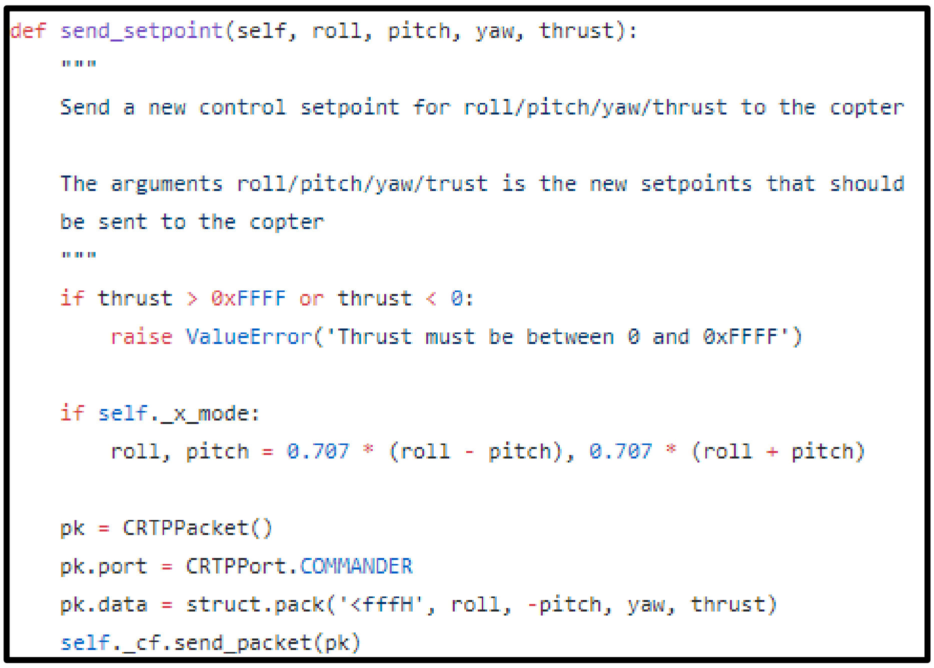
```
"version": 1,
"client_name": "ZMQ client",
"ctrl": {
    "roll": 0.0,
    "pitch": 0.0,
    "yaw": 0.0,
    "thrust": 0.0
```

3. **Sending setpoint commands using the Crazyflie API** – the Crazyflie API allows big 4 variable commands to be sent to the drone using the methods outlined in the python file "Commander.py". Sending commands via this method makes it apply for 500ms before it is cut out by the firmware. In an ideal system 100 commands should be sent in 1 second allowing for very precise control. (1 command every 10ms) (Bitcraze, 2019)
   a. As with ZMQ there is a safety lock in setpoint commands, a command with thrust = 0 must be sent before other values.
   b. The firmware contains a state estimator that can estimate the CrazyFlie's orientation and position using its built-in sensors (Barometer, Gyroscope etc).
   c. **Disadvantage** – Similarly to method 2, this method will also result in a backlog if too many commands are sent that cannot currently be executed (due to commands being sent to quickly). If packets accumulate then proportional latency will be experienced. A work around to this was sending commands followed by a delay to minimize latency created by the cache backlog. This backlog only results in latency and does not cause any fatal errors. So, with an adequately timed loop the backlog can be kept short and the PID can run smoothly. (Bitcraze, 2018)
   d. **[Figure 25, Set point command Implementation]**



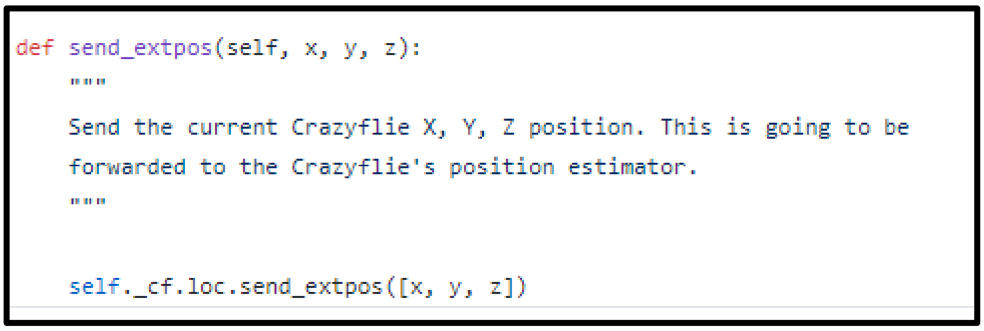
```
roll     = 0.0
pitch    = 0.0
yawrate  = 0
thrust   = 0
crazyflie.commander.send_setpoint(roll, pitch, yawrate, thrust)
```

   e. **[Figure 26, Method within API]**



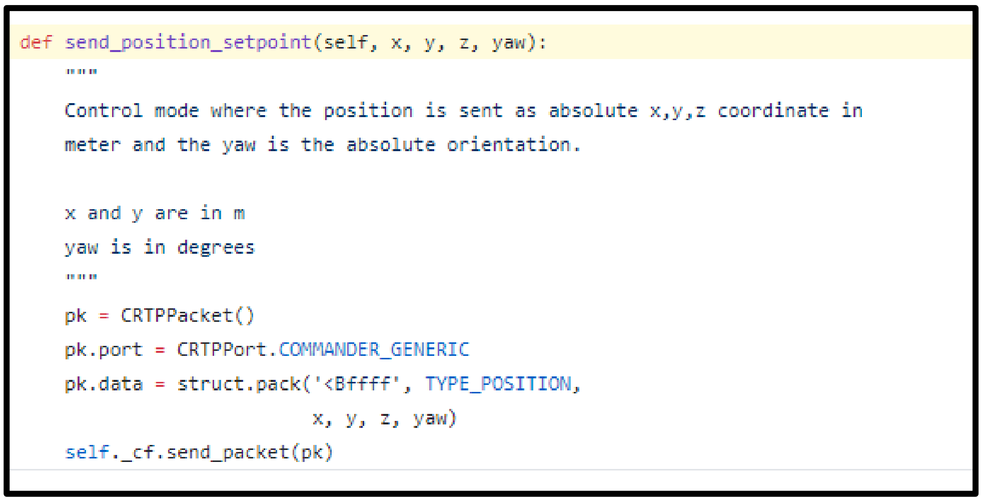
```
def send_setpoint(self, roll, pitch, yaw, thrust):
    """
    Send a new control setpoint for roll/pitch/yaw/thrust to the copter

    The arguments roll/pitch/yaw/trust is the new setpoints that should
    be sent to the copter
    """
    if thrust > 0xFFFF or thrust < 0:
        raise ValueError('Thrust must be between 0 and 0xFFFF')

    if self._x_mode:
        roll, pitch = 0.707 * (roll - pitch), 0.707 * (roll + pitch)

    pk = CRTPPacket()
    pk.port = CRTPPort.COMMANDER
    pk.data = struct.pack('<fffH', roll, -pitch, yaw, thrust)
    self._cf.send_packet(pk)
```

4. **Sending external positional data and 3D positional commands to the drone's firmware using the Crazyflie API –** By making use of multiple methods across various python files within the API (CFLIB) it is possible to send a positional command indicating a specific 3d location for the drone to travel to, this command is supported by a frequent flow of data acquired from a vision system (Vicon Tracker).
    a. this method requires alterations in the CrazyFlie's firmware to instruct the drone to use a different form of estimation. The form of estimation used with this method is a Kalman estimator that makes use of the positional data sent to the drone along with the sensor information it already has (with a barometer onboard it can measure changes in air pressure meaning it is able to determine how high above the ground it is to a certain extent), to produce accurate commands.
    b. This method uses a built in PID controller within the firmware of the drone to regulate thrust so only regular positional information and an initial setpoint command dictating the location to travel to are needed.
    c. **[Figure 27, the external positional position method within API ("Extpos.py")]**



```
def send_extpos(self, x, y, z):
    """
    Send the current Crazyflie X, Y, Z position. This is going to be
    forwarded to the Crazyflie's position estimator.
    """

    self._cf.loc.send_extpos([x, y, z])
```

    d. **[Figure 28, the setpoint command method within API ("Commander.py")]**



```
def send_position_setpoint(self, x, y, z, yaw):
    """
    Control mode where the position is sent as absolute x,y,z coordinate in
    meter and the yaw is the absolute orientation.

    x and y are in m
    yaw is in degrees
    """
    pk = CRTPPacket()
    pk.port = CRTPPort.COMMANDER_GENERIC
    pk.data = struct.pack('<Bffff', TYPE_POSITION,
                          x, y, z, yaw)
    self._cf.send_packet(pk)
```

4.2.2. Discussion on Control commands

Of all methods used in experiments and trails method 3 proved the most fruitful as it was efficient and never crashed due to any faults other than developmental. Its implementation is simplistic and returns appropriate errors if problems arise and when compared to ZMQ commands is the most dependable.

Method 4 was discovered very late into the development cycle and left no room for errors, with erratic 3D flight patterns being recorded even when the simplest commands were input with no explanation or logging available to decipherer the results obtained (locks out all logging/ parameter checking within the code implemented).

The selection of method 3 is based upon its dependability and functionality outweighing that of the first 2 methods however, method 4 could prove to be useful in this project context if further research into it is conducted.

4.3. Radio frequency/Band

ZMQ and setpoint commands (Crazyflie API) which make use of the CrazyRadio Protocol cannot be used simultaneously, for setpoint commands to be sent accurately during program execution it is highly recommended that a higher bandwidth is selected as the default of 250k has a longer range but has a higher chance of collision. (Bitcraze, 2019)

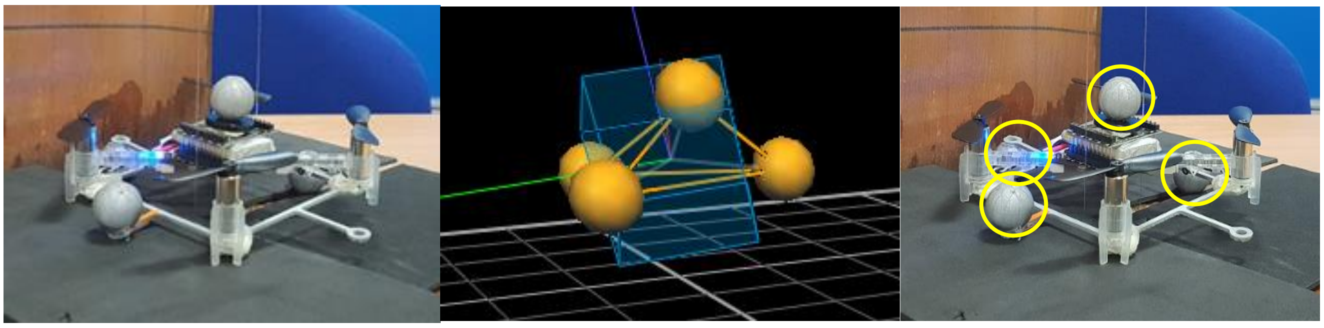
Likewise, it is recommended to change the broadcast address as this will collide with other wireless devices such as WIFI within a small room.

An easy way to tell what your bandwidth is, is by examining your address, the final value of your bandwidth being either 250k, 1m or 2m. Before that is your channel, channels 0-80 are normal use, channels 81-125 are special use and you may be prosecuted if you make use of these.

Original - > **[radio://0/80/250K]**, Used for this project - > **[radio://0/80/2M]**

Note : To alter, first connect to the Crazyflie 2.0 with the normal connect button. Then open “Crazyflie→ Configure 2.0” to reach the configure 2.0 dialog.

4.4. Accessing the Vicon Data Stream


[Figure 29, Representation of CrazyFlie 2.0 Drone markers within Vicon Tracker system, marker configuration Mark 1]

4.4.1. Vicon Tracker Objects

Objects within the tracker system must be created using at least 3 markers, after creation the system will pick up any markers placed within its environment and relate the distance between the markers against all known objects that are being tracked. This allows the system to track multiple objects with minimal confusion as markers are cross referenced with object marker distance information.

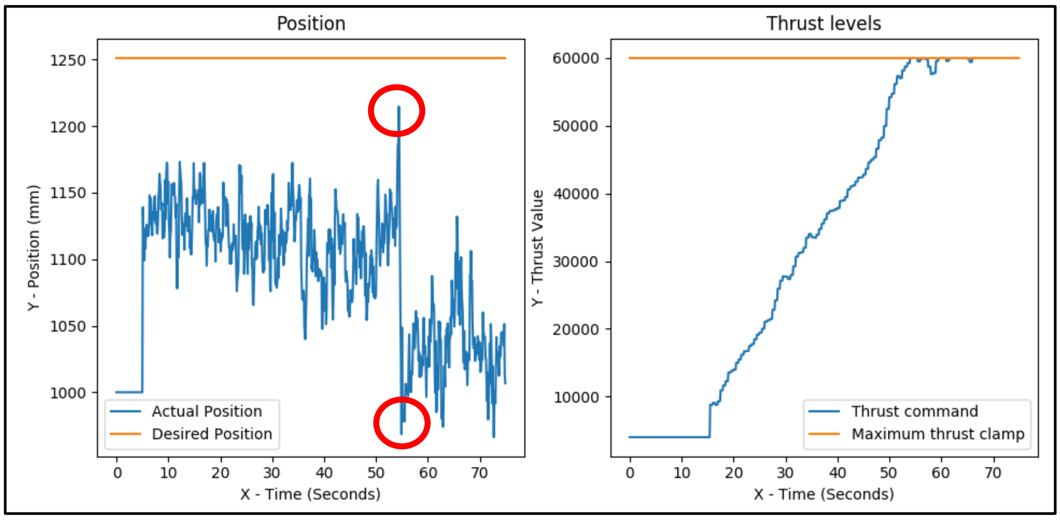
4.4.1.1. Axis Flipping/Swapping

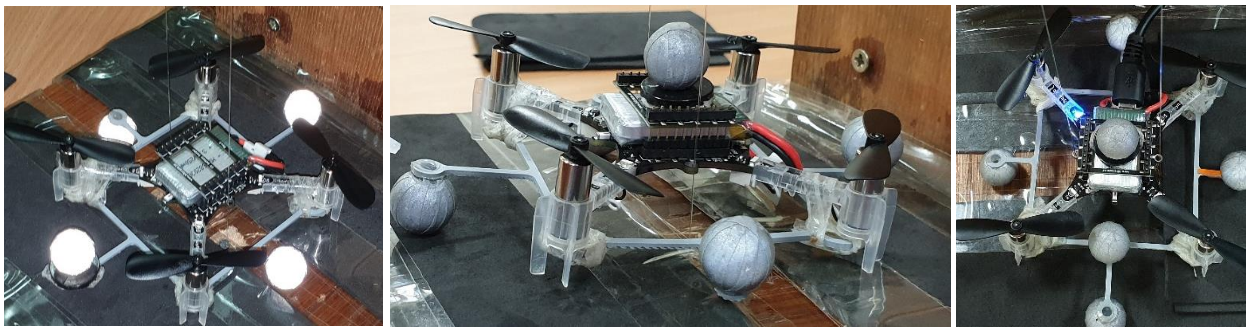
One major problem which was falsely identified in the early stages and lead to major problems in the later stages of the project arose from the configuration of the markers, this led to an issue that was dubbed *Axis Flipping or Axis Swapping*. This issue is caused by symmetric marker layouts that are misinterpreted by the Vicon system, e.g. the system will at times interpret a symmetrical marker configuration object as upside down if there is no specific marker that would indicate the objects preferred origin state (*Axis Flip*) or interpret the Z axis as the X axis (*Axis Swap*) as there is no marker to define the difference . *Axis flipping* is more often the case with the Y axis and *Axis Swapping* is more often the case with the X and Z axis.

An *Axis flip* can be easily identified in the Y axis by a sudden change of +- 200 (Figure 30 below), X and Z aren't as easily identifiable in a scenario where they each respectfully have similar values.

Safeguards against axis swapping / flipping:

- Use unique non-symmetric marker configuration (laws of aerodynamics will conflict causing some drag)
- Implement a function to identify sharp unexpected changes in transitional values (possibly monitor rate of change and standard deviation of Vicon values and then flag values that deviate and have a different rate of change).




[Figure 31, left mark 4, centre mark 3, right mark 5]

4.4.1.2. Configuration of Markers

Mark	1	3	4	5
Observations	Misinterpretation of all axis's once movement is introduced	Stable X and Z but if a single marker becomes obstructed then Y axis is flipped	Surprisingly stable results, but does experience Y axis flipping when movement is introduced	Works perfectly, not a single axis flip/swap experienced due to defined marker positions. The drone struggles but is able to lift the extra weight.
Number of markers	4	5	4	6
Weight added to drone	2 grams	2.5 grams	2 grams	3 grams

[Figure 32, Comparison of different marker configurations, the CrazyFlie 2.0 can support up to 5g of extra weight]

4.4.2. Vicon Tracker Data Stream

Once an object has been created within the Vicon environment the next challenge is to access the positional data which is sent at a rate of 100Hz or 0.01 seconds (This is adjustable, up to 2000hz). Vicon has 3 different recommended methods available to access this data.

The methods to access data from within the Vicon tracker systems data stream (which in this instance will act as a server, sending data outwards) are either via the use of the VRPN service, UDP packets or the companies very own SDK. Each of these methods were investigated.

4.4.2.1. VRPN

The Virtual Reality Peripheral Network is a library that is part of the public domain create by Russell M. Taylor at the University of North Carolina. (Further details on this route of data stream manipulating can be found in appendix D). VRPN was not selected after issues between Python version compatibility arose during installation.

4.4.2.2. Vicon DataStream SDK

Libraries used to receive the data sent out by the Vicon server coded in C++. Use of the SDK supplied by Vicon to access their data stream at first seemed like the logical approach, as it would have granted the client access to all functions and methods written in the Vicon API to access the camera data. However, attempts at binding the C++ functions into Python proved difficult which then implied a separate C++ client would be needed simply for receiving/sending of data which seemed redundant, so the Vicon DataStream SDK was not selected.

- **Most Data / Over complicated Implementation**
  - https://www.vicon.com/products/software/datastream-sdk

4.4.2.3. UDP

User Datagram Protocol (UDP) is a low-latency and loss-tolerating data transportation method used to communicate in networks, it is a member of most socket libraries. It is a connectionless protocol which is described as not needing any initial connection prior to sending or receiving to set up data transfer relationship.

- **Least data / fast access times**

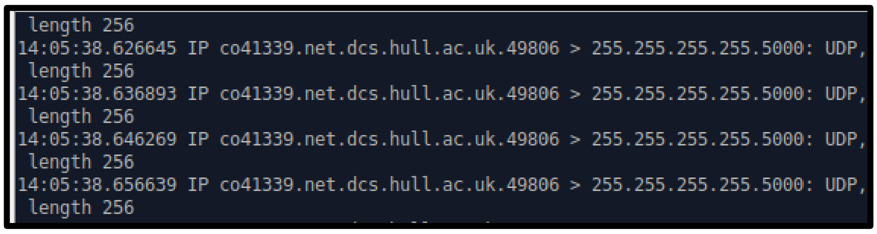
One clear distinction to make between the methods of sending data used within this project is that the data sent to the drone (using method 2) is via ZMQ which uses a TCP/IP connection type or via the Crazyflie API (Method 3), these are what is referred to as connection-oriented, as a connection must first be established prior to sending and receiving data. On the other hand, UDP is a connectionless protocol which is described as not needing any initial connection prior to sending or receiving to set up protocols.

TCP implements stream protocols that allow you to read a vast range of different sized chunks, UDP implements a message protocol that limits datagrams to a certain buffer size (e.g. 256, 512, 1024)

One issue with using UDP in conjunction with the Vicon Tracker system that was faced, was pre-set addresses for sending data, if there was simply a setting to input your desired IP address to send the data-stream to, this would have made intercepting it easier. As it stood the user would have to be connected to the same network as the Vicon equipment to access its data

- As students do not have the credentials to install any software onto the Green Room PC the initial idea of installing VirtualBox onto the green room pc with a bridged connection to access the network had to be scrapped.
    - Along with ideas of simply creating a relay server to repeat the information to a desired IP addresses (similar reason, students could not install any form of Python or any other as they required admin rights)
    - a basic client/ server was separately developed to fully understand the complexities behind connectionless communication and then implemented into the client.

Notes: The bash command 'sudo tcpdump' is a packet sniffer/package analyser tool that is very helpful for validation of data being received on a network.



```
 length 256
14:05:38.626645 IP co41339.net.dcs.hull.ac.uk.49806 > 255.255.255.255.5000: UDP,
 length 256
14:05:38.636893 IP co41339.net.dcs.hull.ac.uk.49806 > 255.255.255.255.5000: UDP,
 length 256
14:05:38.646269 IP co41339.net.dcs.hull.ac.uk.49806 > 255.255.255.255.5000: UDP,
 length 256
14:05:38.656639 IP co41339.net.dcs.hull.ac.uk.49806 > 255.255.255.255.5000: UDP,
 length 256
```

[Figure 33, users can view if they are connected to the correct network by using Sudo tcpdump]

```
"b'3\x89.\x00\x01\x00H\x00Glasses02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00{\
x18\xa1_R\xee\x92@l\xeb~\xab:\xc5\x93\xc0KM\x11\xb6&\xc6\x87@4\x91W\r\xfd\xb5\x90?\xdd''?\xaa\
xd6\x90?\x02\xadMI\x99O\x06\xc0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
```

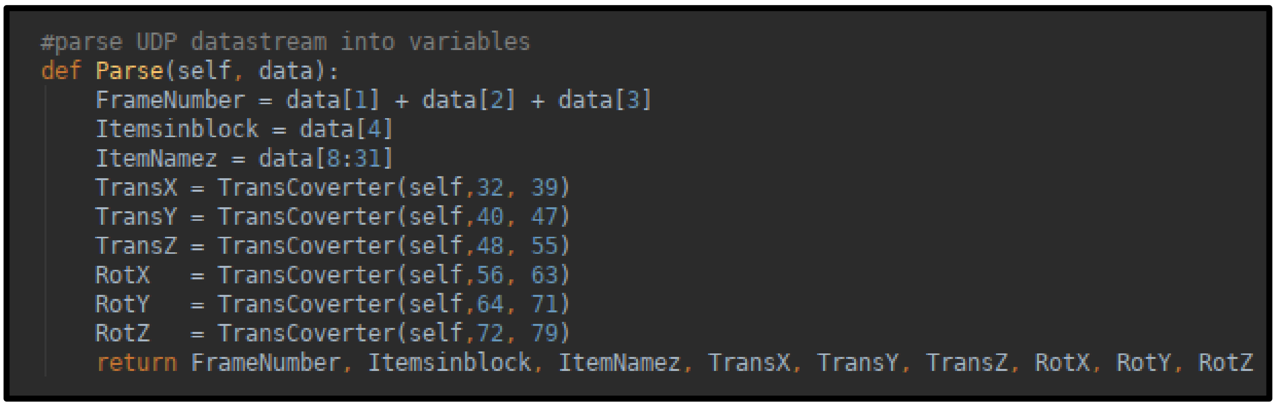
[Figure 34, Raw 256-byte width UDP data from Vicon Broadcast]

4.4.3. Receiving The Stream

To receive and process the raw data shown above, function "ConnectUDP" was created that assigns default values and searches for the selected subnet mask according to what the user has selected within the client, the function group then attempts to bind to the specified address and port. If this is successful, the data stream begins the parsing process where data received will be split and processed according to guidelines. E.g. Bits 56 – 63 contain all the information regarding the Rotation of the object in the X dimension.

The following rules apply to the 256 UDP stream packet (numbers represent what data is stored in the specific byte position of the UDP packet):

- Frame number = [0-3]
- Number of items in block (1 drone or 2 drones) = [4]
- The Objects name = [8-31]
- Transitional movements X = [32-39]
- Transitional movement Y = [40-47]
- Transitional movement Z = [48-55]
- Rotation X = [56-63]
- Rotation Y = [64-71]
- Rotation Z = [72-79]



```
#parse UDP datastream into variables
def Parse(self, data):
    FrameNumber = data[1] + data[2] + data[3]
    Itemsinblock = data[4]
    ItemNamez = data[8:31]
    TransX = TransCoverter(self,32, 39)
    TransY = TransCoverter(self,40, 47)
    TransZ = TransCoverter(self,48, 55)
    RotX   = TransCoverter(self,56, 63)
    RotY   = TransCoverter(self,64, 71)
    RotZ   = TransCoverter(self,72, 79)
    return FrameNumber, Itemsinblock, ItemNamez, TransX, TransY, TransZ, RotX, RotY, RotZ
```

[Figure 35, Method "Parse" will sort the raw data into appropriate variables for each packet received]

4.4.4. Parsing the data

In a linear system, our collected positional information would now need to be parsed into a sensible data format to calculate the crucial error term needed for the PID controller. However, with real world applications, come real world interferences and so is the case with the Vicon Tracker system with regards to varying levels of noise being collected by the sensors.

To minimize the effects of noise on the data, a function was created to invalidate values over/ under a certain threshold (determined by the previous value) as one packet could not have values exceeding those of the previous with regards to a certain negative or positive gain.

- Acceptable - 1, 2, 2, 3, 5, 2, 4, 5, 6
- Unacceptable - 1, 9, 2, 2, 4, 3, 5,-6

Erroneous readings are ignored. (9 and -6 in the above context would be ignored)

4.4.5. Handling multiple packets

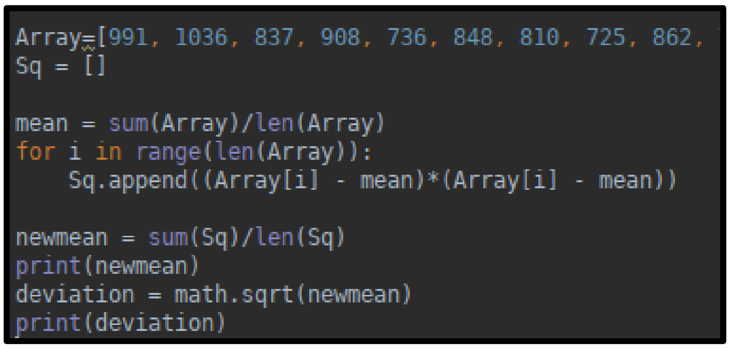
With major noise being eradicated, the next step was to lay out the foundation for the averaging of multiple packet values. The Vicon streams packets at a rate of **100Hz** so for our client to pick up rapid changes over time while taking into consideration those of the past, an efficient algorithm would need to be invested into.

A = [991, 1036, 837, 908, 736, 848, 810, 725, 862, 794, 1002, 938, 1019, 896, 1014, 1144, 867, 791, 741, 696, 762, 783, 805, 751, 675, 847, 1056, 755, 861, 1069, 778, 886, 1102, 1099, 651, 778, 927, 665, 856, 827, 1030, 953, 886, 1015, 722, 1087, 1075, 1060, 753, 980]

B = [1036, 837, 908, 736, 848, 810, 725, 862, 794, 1002, 938, 1019, 896, 1014, 1144, 867, 791, 741, 696, 762, 783, 805, 751, 675, 847, 1056, 755, 861, 1069, 778, 886, 1102, 1099, 651, 778, 927, 665, 856, 827, 1030, 953, 886, 1015, 722, 1087, 1075, 1060, 753, 980, 866]

$$S = \sqrt{\frac{\Sigma (X - \bar{X})^2}{N}}$$

[Figure 36, standard deviation equation]



```
Array=[991, 1036, 837, 908, 736, 848, 810, 725, 862,
Sq = []

mean = sum(Array)/len(Array)
for i in range(len(Array)):
    Sq.append((Array[i] - mean)*(Array[i] - mean))

newmean = sum(Sq)/len(Sq)
print(newmean)
deviation = math.sqrt(newmean)
print(deviation)
```

[figure 37, created code to solve standard deviation]

A has the following properties:

- Length = 50
- Mean = 880
- Standard Deviation = 133.28

B has the following properties:

- Length = 50
- Mean = 882
- Standard Deviation = 132.40

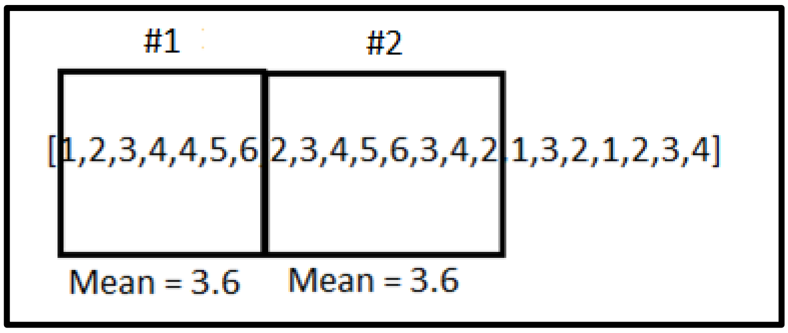
From this we can deduce with some certainty that: **Any value that deviates by more than 260 from the mean should be considered as genuine directional movement from the drone in an examined axis.**

4.4.6. Packet value Averaging techniques

4.4.6.1. Original Averaging of packet Implementation

Initially once examining the data the first approach was implementing a while i < 100 loop to iterate through each of the packets sent to get 1 generalized value for each second for each of the values (100 packets in 1 second). This loop included functions that performed tasks such as dumping erroneous noise data as described above as well as calculating the mean.

This implementation caused issues with error values, as each second the average could be slightly different from the last (E.g. 1)400 and 2)500) even while the drone was stationary. This could have been due to the threshold used (if result is within +- 100 of previous then use), the initialization of the function or issues determining if the first packet received was noisy data that would lead to erroneous packets being preferred over accurate measurements.



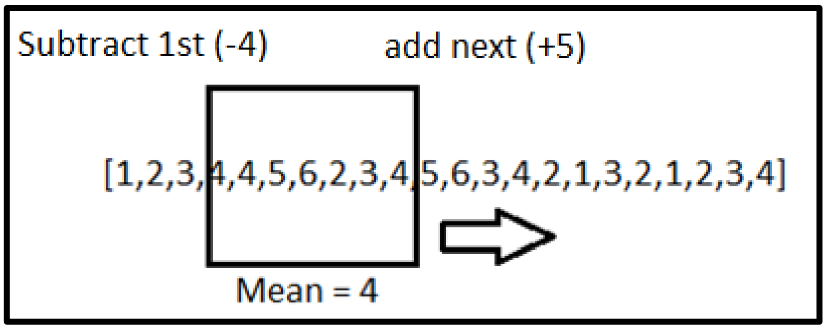
[Figure 38, original implementation for averaging of packets]

[Major Problem] – Noise effecting error term values causing erratic movements from drone

This topic of efficient averaging of packet data was later revisited and code was refactored as inaccurate measurement processing would lead to incorrect error term results which would cause chaos later within the PID controller.

4.4.6.2. Sliding Window

After many complications with noisy stream data, a dependable and efficient averaging algorithm was researched and implemented. This averaging technic gradually “slides” over the data segment for which it is currently averaging. In operation a programmer must provide a sample space, the algorithm then takes a window of this sample space and averages it. The window then moves to the right and the first record of data that was stored is deleted and a new record is added. This process is repeated until the window reaches the end of the sample space.



[figure 39, Representing Sliding window functionality, method gets a mean of a specific “window” of data and then slides over to the next item in the sample space, this deletes the value furthest to the left and adds the next item outside of the window]

```
Windowsize = 50
ArrayTransX = []
AverageTransX = 0
Loop
    TotalX = []
        While len (TotalX)< windowsize
            Data = Get Vicon data
            TransX = Parse (Data)
            If len (ArrayTransX) < windowsize
                AverageTransX += TransX
                ArrayTransX.append(TransX)
            Else:
                TotalX.append(AverageTransX)
                AverageTransX, ArrayTransX = SlidingWindow(AverageTransX, ArrayTransX)
        ActualAverageX = sum(TotalX) / len(TotalX) / windowsize
Print (ActualAverageX)
```

[Figure 40, Process of obtaining and averaging X axis values from the Vicon system]

Note: Traditional Sliding window techniques make use of 0 padded sliding window averages if the window size is to small (e.g. a window of size 10 with only 1 value of 50 in it equals a mean of 5). The algorithm implemented in this project waits for 50 values (window size) before it begins the sliding window due to inaccuracies relating to the Vicon system always sending lower end deviation values on initial setup of the received stream. (if actual x= 30 then Vicon sends measured x=25 initially)

4.5. PID Implementation

The most meaningful developmental decision that had to be made within the latter stage of the project was whether to implement the PID control loop within the Client or within the firmware running on the CrazyFlie 2.0 drone.

4.5.1. Client vs Firmware PID Implementation

	Client	**Firmware**	**Choice**
Computational processing speed of commands	**Higher** due to dynamic hardware advantages of the Virtual machine. (Processor and Ram can be altered according to hardware available)	**Lower** due to minimal hardware on the device. (Cortex-M4 Processor @168MHz / SRam 192kb)	**Client**
Successful execution rate of commands	**Low**, ZMQ injections can frequently result in lost packets of data. **Mid-High,** commands using the CrazyFlie API have a higher success rate but do result in occasional packet loss	**Low**, Method 4 command types proved erratic.	**Client**
Amount of data sent to CrazyFlie	**Lower**, with all UDP stream data parsed and propeller speed changes calculated within a client based PID, only the crucial big 4 variables need to be altered (Thrust, Pitch, Yaw, Roll) and sent to the drone.	**Higher,** initially a desired set point position must be sent and then external positional information (X, Y, Z) as well as orientation (Yaw) must be sent to the drone frequently for the internal PID to make appropriate alterations to variables. .	**Client**
Language	**Python**	**C/C++**	**Client**
PID Controller implementation will be on the :		**Client**	

[Figure 41, Client vs firmware PID comparison]

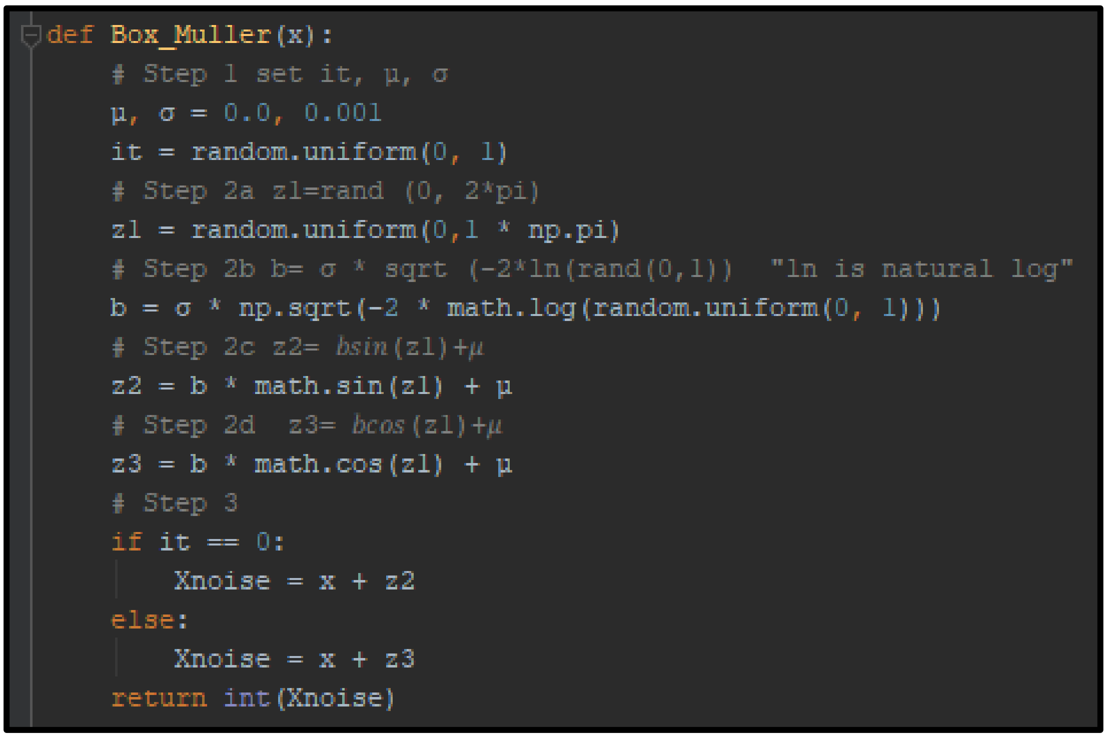
4.5.2. The PID Controller

```
previous_error = 0
integral = 0
loop:
  error = setpoint - measured_value
  integral = integral + error * dt
  derivative = (error - previous_error) / dt
  output = Kp * error + Ki * integral + Kd * derivative
  previous_error = error
  wait(dt)
  goto loop
```

[Figure 42, Pseudo code of the methods used to calculate the output of the PID controller(Thrust in this project) (Icady, 2009)**]**

4.5.3. PID Simulations

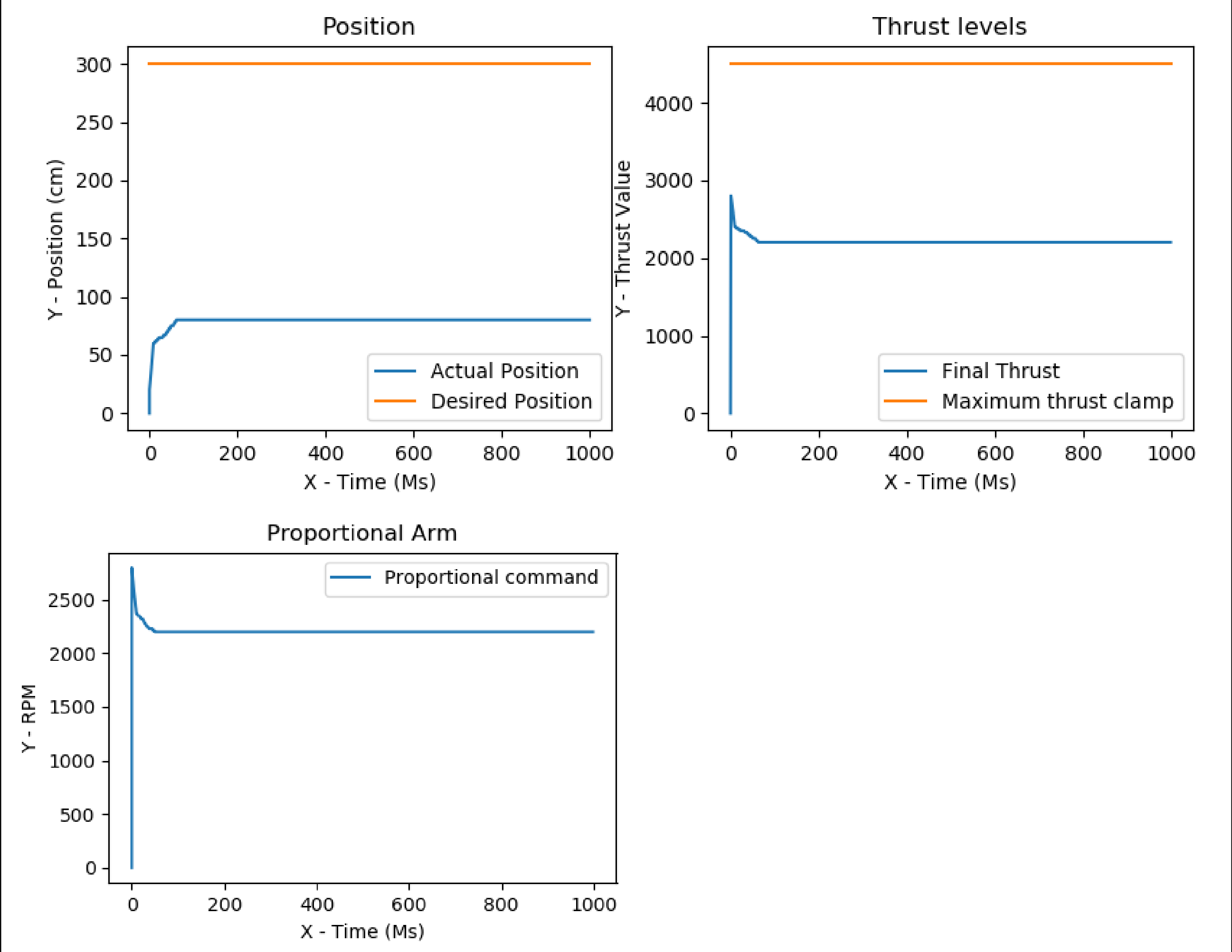
To bypass the client overhead, test results quickly with minimal risk to equipment and to improve the functionality of the PID controller, a simulation system was developed. The controller developed in this system is not limited to quadcopter thrust regulation and can easily be integrated into other systems in need of a regulator. A distinction should be made between the results bellow which are strictly simulations and the results obtained later in the report that make use of the Vicon system as feedback. The simulation uses a Box Muller method to introduce noise into positional results to imitate real world noise which is present in the Vicon system.



```
def Box_Muller(x):
    # Step 1 set it, µ, σ
    µ, σ = 0.0, 0.001
    it = random.uniform(0, 1)
    # Step 2a z1=rand (0, 2*pi)
    z1 = random.uniform(0,1 * np.pi)
    # Step 2b b= σ * sqrt (-2*ln(rand(0,1))  "ln is natural log"
    b = σ * np.sqrt(-2 * math.log(random.uniform(0, 1)))
    # Step 2c z2= bsin(z1)+µ
    z2 = b * math.sin(z1) + µ
    # Step 2d  z3= bcos(z1)+µ
    z3 = b * math.cos(z1) + µ
    # Step 3
    if it == 0:
        Xnoise = x + z2
    else:
        Xnoise = x + z3
    return int(Xnoise)
```

[Figure 43, Code within the PID simulation to add noise to positional results to mimic that of a real-world system]

4.5.3.1. Proportional Arm observations



[Figure 44, represents early results of the controller using only the proportional (Kp) arm (noise added) and a request to rise to 300 cm, it shows signs of steady state error]

The following results are obtained by only using the proportional arm of the controller.
Our Proportional arm is given the following parameters:

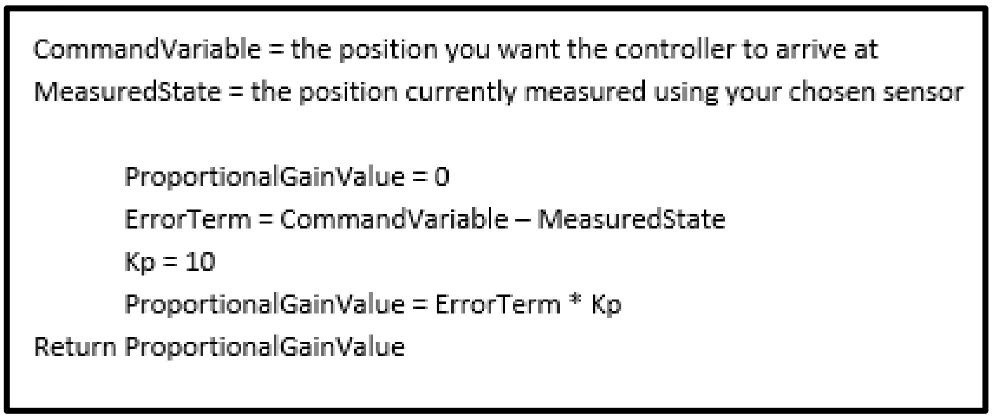
- Kp = 10
- command variable = 300
- current position = 0
- error = command variable – current position
- Noise added

$$u(t) = K_p e(t)$$

Initially the thrust levels using the above values increase by a notable amount as ***Thrust = error * Kp***.

What can we deduce from this information?

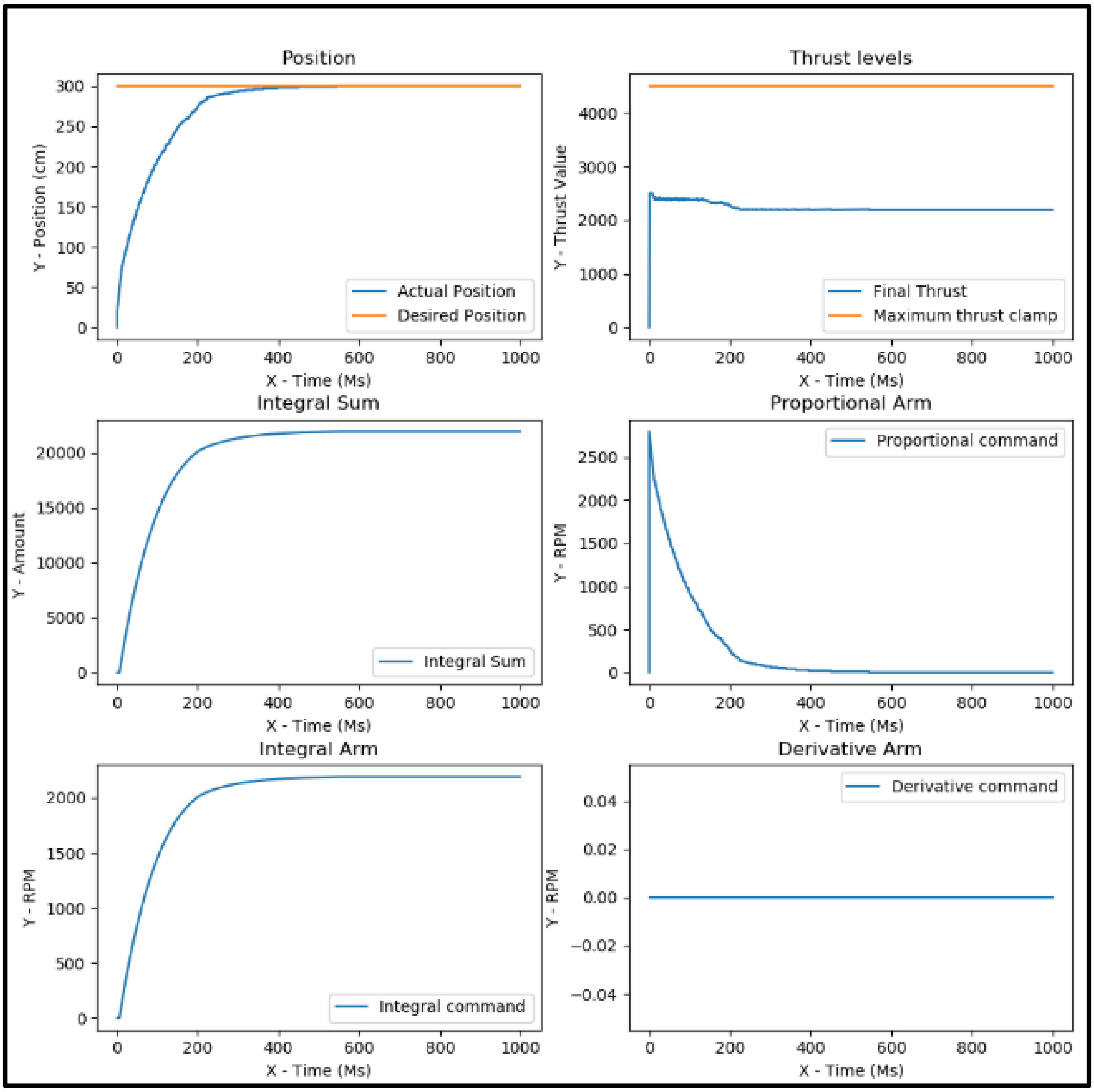
- As error approaches 0, thrust levels decrease in size.
- If error remains constant, then thrust remains constant using only the Kp arm.
- If constant error and constant Kp then there will be a constant thrust
- Steady state error is identified by an inability to change position using a constant Kp as error remains unchanged and Kp is a constant the result will remain constant.



```
CommandVariable = the position you want the controller to arrive at
MeasuredState = the position currently measured using your chosen sensor

        ProportionalGainValue = 0
        ErrorTerm = CommandVariable – MeasuredState
        Kp = 10
        ProportionalGainValue = ErrorTerm * Kp
Return ProportionalGainValue
```

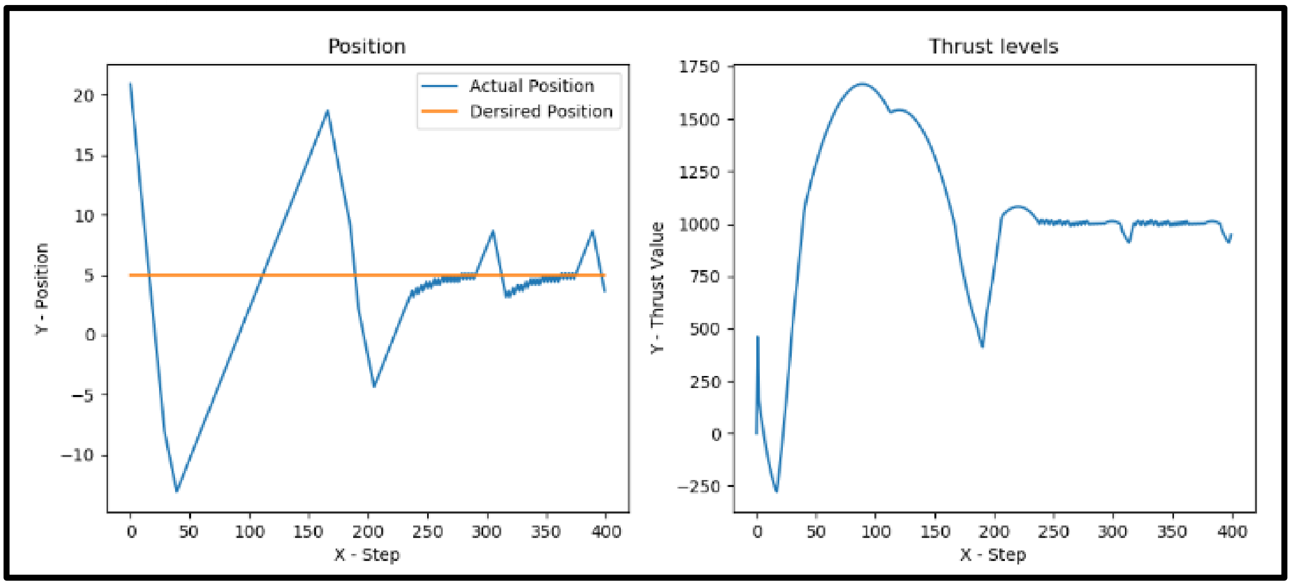
[Figure 45, Pseudo code for the Proportional arm]

4.5.3.2. Integral Arm observations



[Figure 46, Represents the PI controller with D turned off (Noise added and clamp active to prevent overshoot)]

$$u(t) = K_{\mathrm{p}} e(t) + K_{\mathrm{i}} \int_0^t e(t')\,dt'$$



[Figure 47, represents early stages of the PI controlled thrust traveling in 1 dimension, exhibiting what is known as overshooting (no noise and no clamp)]

Of the three arms used to create the PID, the integral was the most complex to implement and has the greatest effect on the output when Ki = Kd = Kp (say 2 = 2 = 2). The integral arm keeps a running sum of all the error terms. In a case where the controller has arrived at its command, the integral sum is still outputting values causing the drone to carry on rising, negative error terms then enter the integral sum once the command is passed reducing the thrust value rapidly until the drone begins to change position (evident in figure 45). This isolation or zig zag shape is known as overshooting and it can be prevented using a clamp that informs the Integrator to stop adding values to the integral sum once the sum of all arms creates a thrust value that leads to an error of 0. (It sets the value to be added to the integral sum to 0 to stop growth [figure 44])

Properties of the Integral arm:

- Kp = 10, Ki = 0.1
- Command variable (figure) = 300
- Keeps a running total.
- Determines if a value is saturating and caps thrust if necessary.
- Caps thrust at a constant value which is represented by the systems near maximum output value (values over this are saturating (Cannot possibly be achieved)).
- Checks if thrust and error term are the same sign .
- If signs are the same and saturating, then clamp.
- If clamp is activated, then an error of 0 is added to the integral sum keeping it at its current state until the controller decides to stop clamping.
- As the integral sum increases, the value output by its arm increases which will eventually reduce error to 0. Once the drone has an error of 0 only the integral arm is outputting values to keep it at its current position with Kp arm outputting 0 as Kp* error term (0) = 0.

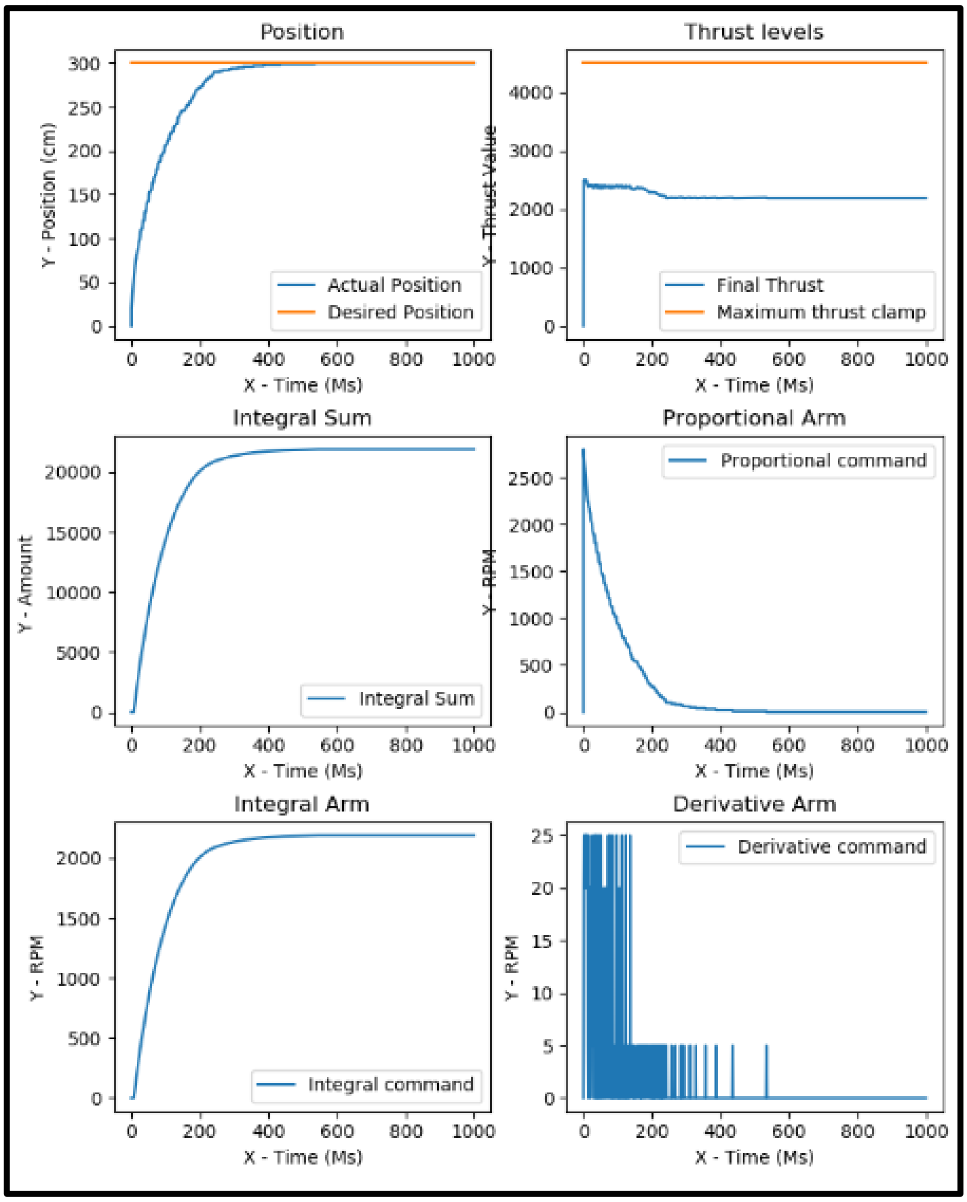
```
CommandVariable = the position you want the controller to arrive at
MeasuredState = the position currently measured using your chosen sensor
IntegralSum = Current sum of all errors accumulated in the system

    IntegralGainValue = 0
    ErrorTerm = CommandVariable – MeasuredState
    Ki = 0.1

    IntegralSum +=  ErrorTerm
    IntegralGainValue = IntegralSum * Ki
Return IntegralGainValue
```

[Figure 48, Pseudo code for the Integral arm without clamp]

```
CommandVariable = the position you want the controller to arrive at
MeasuredState = the position currently measured using your chosen sensor
IntegralSum = Current sum of all errors accumulated in the system
Thrust = Current thrust levels
Clamped = Determines if values should be added to the integral sum or not
ActuatorLimit = the limit of your system (In a drone, the max RPM of the propellers)

    IntegralGainValue = 0
    ErrorTerm = CommandVariable – MeasuredState
    Ki = 0.1

    Saturating, SignsEqual = False, False
    IntegralGainValue = (IntegralSum + ErrorTerm) * Ki
    PreThrust = Thrust + IntegralGainValue
    CurrentThrust = PreThrust
        If PreThrust >= ActuatorLimit
            PreThrust = ActuatorLimit
        If CurrentThrust != PreThrust
            Saturating = True
        If PreThrust and ErrorTerm are both > 0 or PreThrust and ErrorTerm both < 0
            SignsEqual = True
        If SignsEqual and Saturating :
            Do not clamp
        Else
            Clamp
    If Clamp:
        IntegralSum += ErrorTerm
        Thrust += IntegralGainValue
    Else:
        IntegralGainValue = IntegralSum* Ki
        Thrust = PreThrust

Return IntegralGainValue
```

[Figure 49, Pseudo code for the Integral arm with clamp]

4.5.3.3. Derivative Arm observations

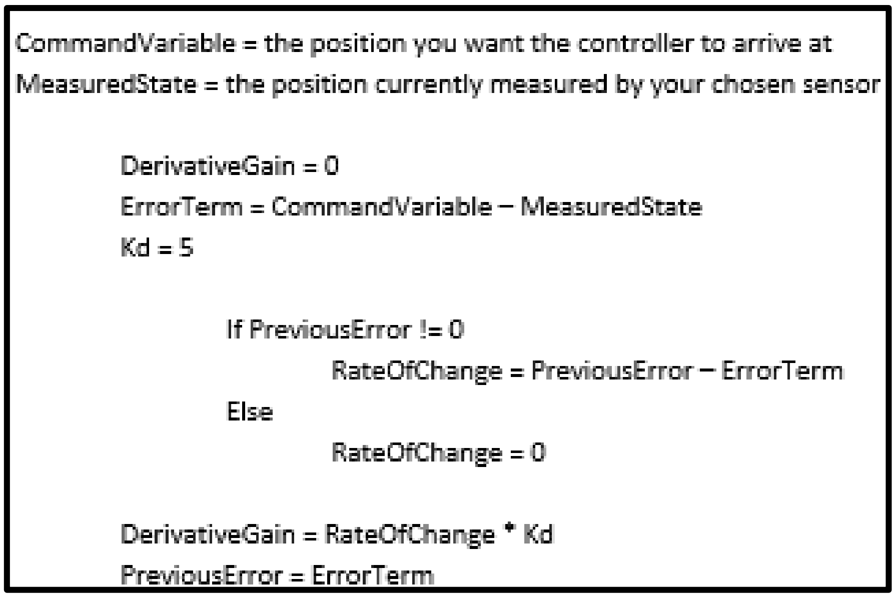


[Figure 50 represents the 3 arms of the PID controller values when requested to regulate thrust to a position of 300cm]

$$u(t) = K_p e(t) + K_i \int_0^t e(t')\,dt' + K_d \frac{de(t)}{dt},$$

The derivative arm influences system actions by predicting the next error term using the rate of change which is obtained by calculating the difference between the current and previous error. The derivative arms main purpose is to reduce the rate of change as the controller approaches the goal and then to assist in subtracting values to ensure the integrator sum does not cause the controller to overshoot as it accumulates (in systems with no clamp).

- Kp = 10, Ki = 0.1 Kd = 5, Command variable = 300
- Increases overall gain of the controller when large error terms are present.
- Decreases overall gain of the controller when smaller error terms are present.
- Similarly, with Kp, Kd has no effect on the system when an error of 0 achieves a stable result (only integrator).



```
CommandVariable = the position you want the controller to arrive at
MeasuredState = the position currently measured by your chosen sensor

        DerivativeGain = 0
        ErrorTerm = CommandVariable – MeasuredState
        Kd = 5

                If PreviousError != 0
                        RateOfChange = PreviousError – ErrorTerm
                Else
                        RateOfChange = 0

        DerivativeGain = RateOfChange * Kd
        PreviousError = ErrorTerm
```

[Figure 51, Pseudo code for the Derivative arm]

4.5.4. PID Simulation Conclusion

A PI controller is sufficient for most tasks that would require a regulator, use of the final arm improves the performance of the controller as whole but does add a level of difficulty in implementation that delivers a minuscule result (in regard to improvement) when comparing a PI controller to a PID controller.

4.6. Overall system development

4.6.1. Command execution speed

As discussed in the *Controlling the CrazyFlie* Section, commands sent to the CrazyFlie are applied for a total of 500ms after which the watchdog will terminate the command resulting in either the next command in the backlog replacing it or inactivity if no command is sent.

If more than 1 command is sent every 500ms/0.5 seconds it will create an ever-increasing sized backlog which results in a perceived latency as commands are executed later than expected. If the time space between sending commands is to large, the drone will enter a state of inactivity dependant on how large the time space is and will mostly result in a freefall.

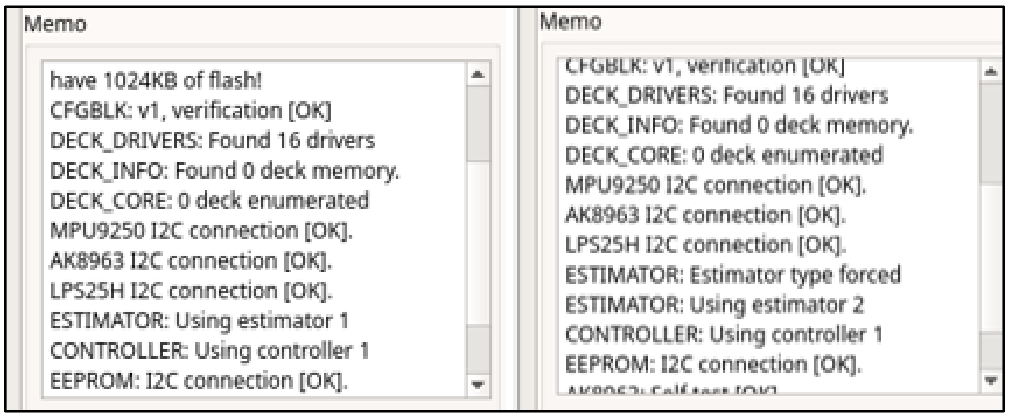
Timing of commands is crucial. Once the backlog reaches a certain length due to excess commands sent while also processing all the needed positional data, it will cause the system to throw an error (Threading [Python 3.6])

For each 10 seconds the system will request 20 positional averages from the sliding window method, 0.5 seconds = 1 Positional average, 1 command every 0.5 seconds.

4.6.2. Switching between command type

Zmq, firmware PID and Crazyflie API setpoint commands cannot be used simultaneously or even sequentially without altering code outside of the client.

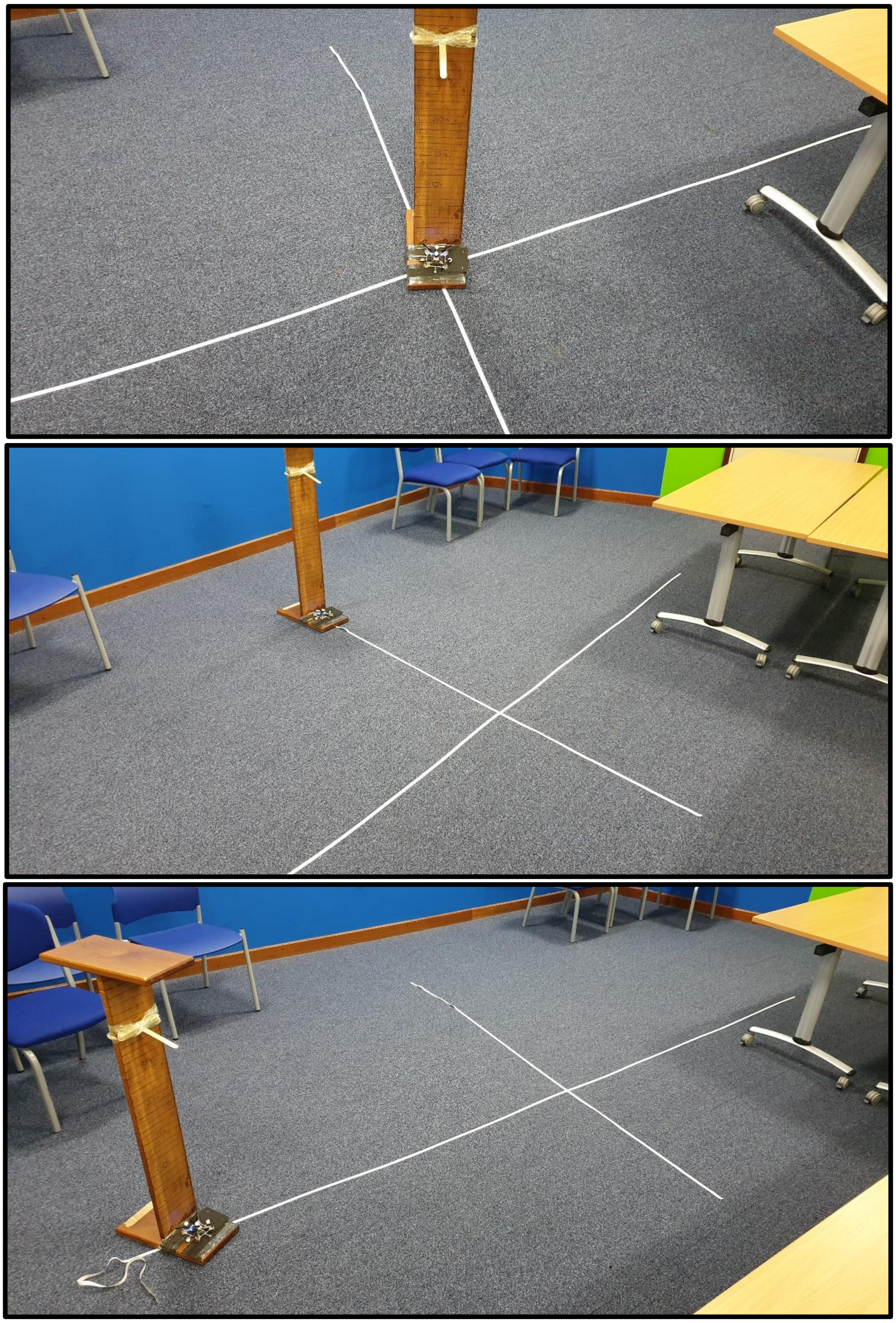
- Zmq requires a change in input device to ZMQ@127.0.0.1:1212 and Zmq to be enabled within a config file located at '/home/bitcraze/projects/crazyflie-clients-python/src/cfclient/configs/config.json' to enable ZMQ (View appendix A.2)
- Use of the firmware PID requires a change in estimators used by the drone (use Kalman estimator). This change requires a firmware flash. File Config.mk.example must be altered to include the line "ESTIMATOR=Kalman" and renamed to Config.mk to switch estimators.
- Setpoint commands require the estimator to be set to default (i.e. cannot be Kalman). This change requires a firmware flash.



```
have 1024KB of flash!
CFGBLK: v1, verification [OK]
DECK_DRIVERS: Found 16 drivers
DECK_INFO: Found 0 deck memory.
DECK_CORE: 0 deck enumerated
MPU9250 I2C connection [OK].
AK8963 I2C connection [OK].
LPS25H I2C connection [OK].
ESTIMATOR: Using estimator 1
CONTROLLER: Using controller 1
EEPROM: I2C connection [OK].
```

```
CFGBLK: v1, verification [OK]
DECK_DRIVERS: Found 16 drivers
DECK_INFO: Found 0 deck memory.
DECK_CORE: 0 deck enumerated
MPU9250 I2C connection [OK].
AK8963 I2C connection [OK].
LPS25H I2C connection [OK].
ESTIMATOR: Estimator type forced
ESTIMATOR: Using estimator 2
CONTROLLER: Using controller 1
EEPROM: I2C connection [OK].
```

[Figure 52, Users can identify which estimator they are currently using through use of the client, Estimator 2 = Kalman]

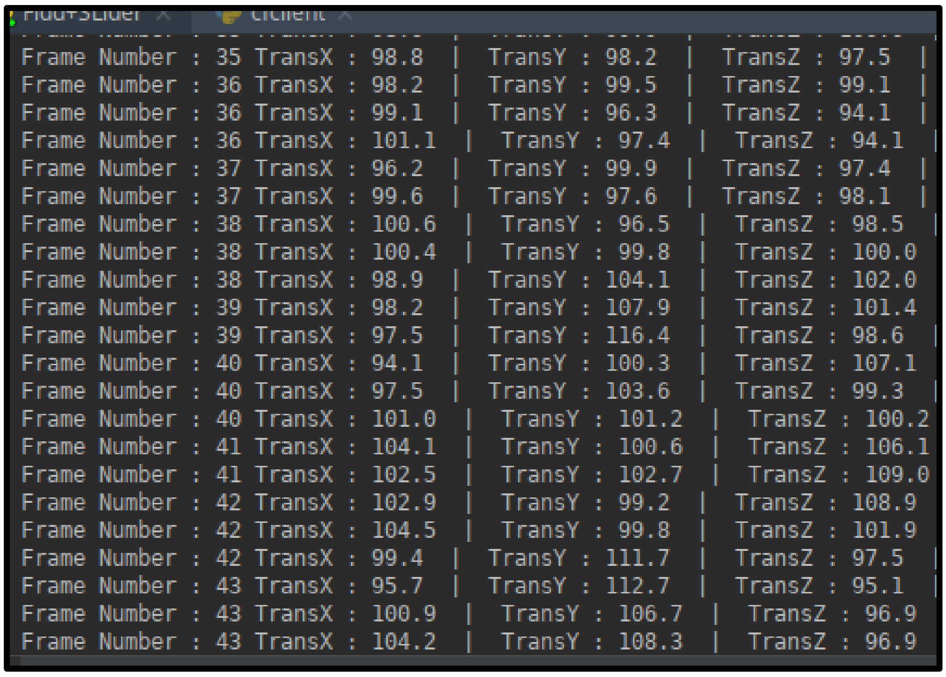


[Figure 53, The following page pictures an experiment conducted which eventually led to the discovery of the *axis swapping / flipping* issues.]

4.6.3. Final Experiments and evaluation of system segments

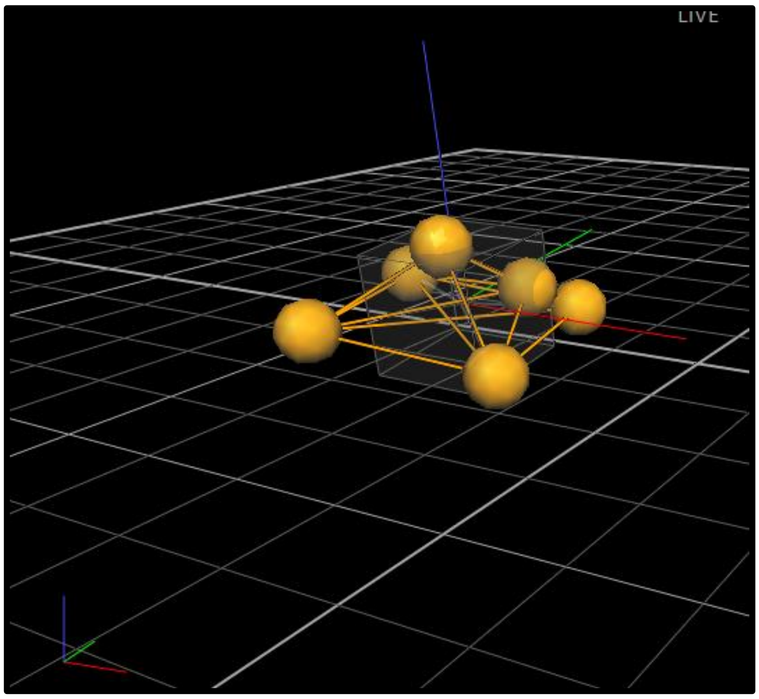
4.6.3.1. Vicon Packets

Each UDP packet from the Vicon system is parsed and averaged using the sliding window technique to efficiently produce semi-accurately averaged results according to what is being received. The sliding window average is altered each time a new value is received from the UDP stream (every 0.01 seconds). The averaging begins once there are 50 transitional data values, they are all used to get an average value for that time space and displayed. This process then continues in the expected manner of a sliding window algorithm and uses every 50th value as the measured positional state.



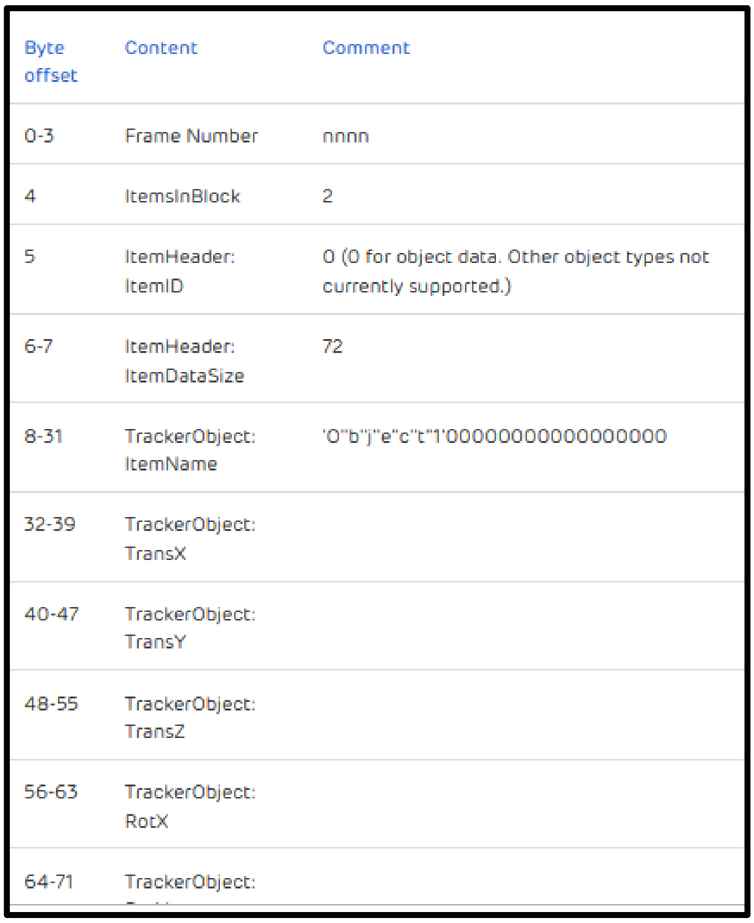
```
Frame Number : 35 TransX : 98.8  |  TransY : 98.2  |  TransZ : 97.5  |
Frame Number : 36 TransX : 98.2  |  TransY : 99.5  |  TransZ : 99.1  |
Frame Number : 36 TransX : 99.1  |  TransY : 96.3  |  TransZ : 94.1  |
Frame Number : 36 TransX : 101.1  |  TransY : 97.4  |  TransZ : 94.1  |
Frame Number : 37 TransX : 96.2  |  TransY : 99.9  |  TransZ : 97.4  |
Frame Number : 37 TransX : 99.6  |  TransY : 97.6  |  TransZ : 98.1  |
Frame Number : 38 TransX : 100.6  |  TransY : 96.5  |  TransZ : 98.5  |
Frame Number : 38 TransX : 100.4  |  TransY : 99.8  |  TransZ : 100.0
Frame Number : 38 TransX : 98.9  |  TransY : 104.1  |  TransZ : 102.0
Frame Number : 39 TransX : 98.2  |  TransY : 107.9  |  TransZ : 101.4
Frame Number : 39 TransX : 97.5  |  TransY : 116.4  |  TransZ : 98.6  |
Frame Number : 40 TransX : 94.1  |  TransY : 100.3  |  TransZ : 107.1
Frame Number : 40 TransX : 97.5  |  TransY : 103.6  |  TransZ : 99.3  |
Frame Number : 40 TransX : 101.0  |  TransY : 101.2  |  TransZ : 100.2
Frame Number : 41 TransX : 104.1  |  TransY : 100.6  |  TransZ : 106.1
Frame Number : 41 TransX : 102.5  |  TransY : 102.7  |  TransZ : 109.0
Frame Number : 42 TransX : 102.9  |  TransY : 99.2  |  TransZ : 108.9
Frame Number : 42 TransX : 104.5  |  TransY : 99.8  |  TransZ : 101.9
Frame Number : 42 TransX : 99.4  |  TransY : 111.7  |  TransZ : 97.5  |
Frame Number : 43 TransX : 95.7  |  TransY : 112.7  |  TransZ : 95.1  |
Frame Number : 43 TransX : 100.9  |  TransY : 106.7  |  TransZ : 96.9
Frame Number : 43 TransX : 104.2  |  TransY : 108.3  |  TransZ : 96.9
```

[Figure 54, Positional information, the drone travelled upwards from frame 38-39 then descended]



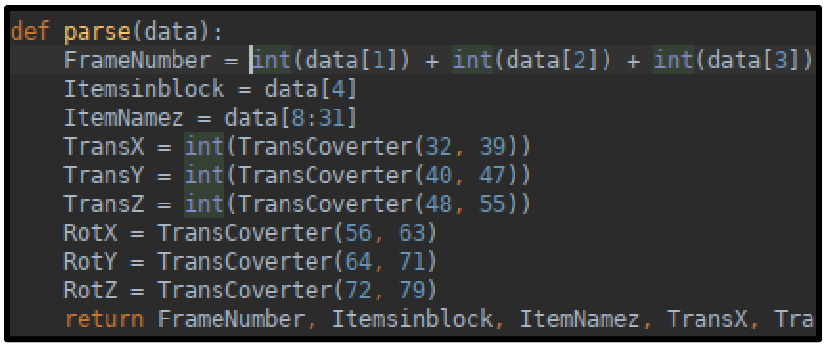
[Figure 55, indicates that the axis of the object is matched with the axis of the Vicon system]

The parsing method follows the Vicon user manual rules for UDP so values being parsed into incorrect variables is not possible in any use case other than Axis swapping/ Flipping.



Byte offset	Content	Comment
0-3	Frame Number	nnnn
4	ItemsInBlock	2
5	ItemHeader: ItemID	0 (0 for object data. Other object types not currently supported.)
6-7	ItemHeader: ItemDataSize	72
8-31	TrackerObject: ItemName	'O''b''j''e''c''t''1'00000000000000000
32-39	TrackerObject: TransX	
40-47	TrackerObject: TransY	
48-55	TrackerObject: TransZ	
56-63	TrackerObject: RotX	
64-71	TrackerObject:	

[Figure 56, Vicon User Guide page 174 UDP packet (Limited, 2018)**]**



```
def parse(data):
    FrameNumber = int(data[1]) + int(data[2]) + int(data[3])
    Itemsinblock = data[4]
    ItemNamez = data[8:31]
    TransX = int(TransCoverter(32, 39))
    TransY = int(TransCoverter(40, 47))
    TransZ = int(TransCoverter(48, 55))
    RotX = TransCoverter(56, 63)
    RotY = TransCoverter(64, 71)
    RotZ = TransCoverter(72, 79)
    return FrameNumber, Itemsinblock, ItemNamez, TransX, Tra
```

[Figure 57, Client parse method]

Axis swapping / Flipping is not possible with the current configuration (Mark 5) as markers have been used to clearly define each axis and the object window within the Vicon tracker system is constantly being monitored for this issue.

Errors are sometimes present at start-up due to configuration issues within the Vicon Tracker environment, realigning axis's will often fix these issues for the Y axis.

4.6.3.2. PID CONTROLLER

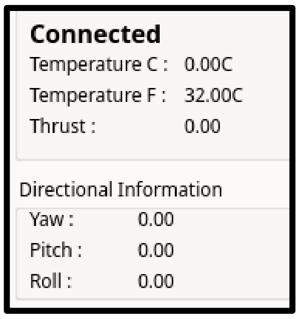
The PID controller performs all functions outlined in the background to an excellent standard, however due to occasional inaccuracies in feedback data this causes spikes to appear. A steadier level of flight can be achieved by decreasing the value of the integral arm significantly (from 3 to 1) but this change causes the growth of the systems control variable to take 2/3 times longer, so any height that would have taken 30 seconds to achieve would now take 90 seconds.

Final PID values:

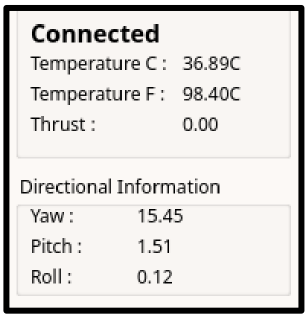
- Kp = 50
- Ki = 3
- Kd = 15

4.6.3.3. CRAZYFLIE DRONE

In the final stages of the project I was unable to make any more significant progress into single dimensional autonmous trajectory as I was being affect by a new error within the Crazyflie library that causes a deadlock within the client making it unable to send any commands to the crazyflie. It is very easily identifible by the inablity to update the logging components within the client on connect [Figure bellow]. This error is not contained to only the modified Crazyflie Client created for this project and persisits within the original bitcraze released version as well



```
Connected
Temperature C :   0.00C
Temperature F :   32.00C
Thrust :          0.00

Directional Information
Yaw :        0.00
Pitch :      0.00
Roll :       0.00
```



```
Connected
Temperature C :   36.89C
Temperature F :   98.40C
Thrust :          0.00

Directional Information
Yaw :        15.45
Pitch :      1.51
Roll :       0.12
```

.

[{Figure 58 left , Deadlock} {Figure 59 right, normal operation}]

One of the lead developers from the Bitcraze project gives this description of the issue : "When reconnecting a Crazflie that has already a couple of log block setup, it can happen that the connection process freezes. This seems to be a dead-lock in the semaphore that protects the protocol version getter when the version is being fetched. One solution is to remove the semaphore and make sure the version is fetched before initializing anything during the Crazyflie connection."

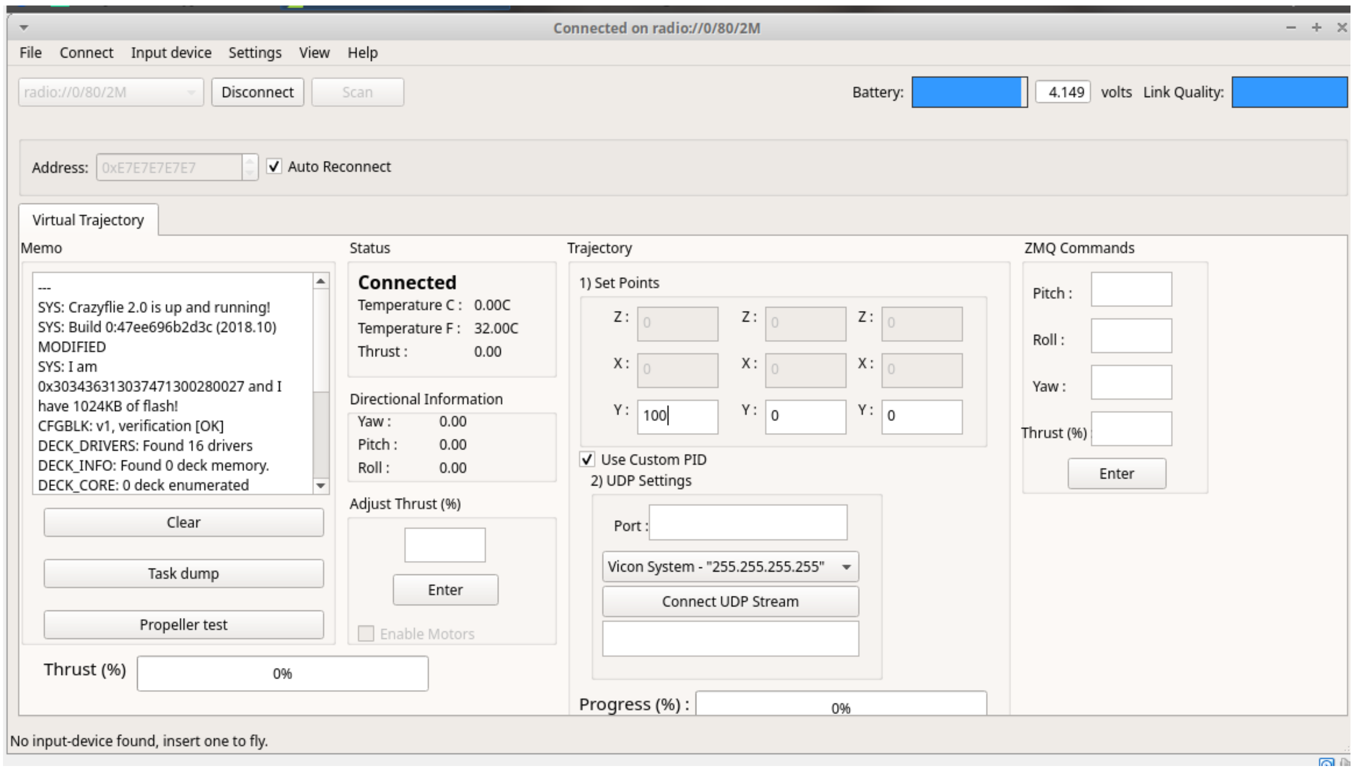
The Comment section regarding this bug can be found here : https://github.com/bitcraze/crazyflie-lib-python/issues/99

Occasionally the drone would continue logging and would accept commands again but due to another error with the firmware of the drone (as of 23/04/2019 running the latest version/ multiple versions used) the yaw value is constantly decreasing. This should not affect the system as it sends a command to set the yaw value to one that has been specified within the code, however in operation this is still visibly present which is identified by the drone leaning towards one direction (In the case of yaw, left or right) originally the work around for this error was simply restarting the Crazyflie to reinitialize the values to 0 and everything would work as expected but currently when you reset the device it renters the issue outlined above regarding the deadlock.

So as one issue resolves another is intensified that can only be fixed through a restart which will trigger the first.

Unfortunately, I was unable to pin point this deadlock as a CFLIB issue because an error is not displayed within the console when this deadlock is reached, and other system functions operate normally other than logging and sending commands, so it was initially assumed to be a self-created coding error until research proved otherwise. As commands could no longer be sent, development stagnated and an impass was reached.

With all of that said the code is able to make the drone rise to a user defined position by stating how much higher you want the drone to rise in relation to its current position (e.g. 100mm will cause the drone at a starting position of 800mm to rise to 900mm)


Connected on radio://0/80/2M

File Connect Input device Settings View Help

radio://0/80/2M Disconnect Scan Battery: 4.149 volts Link Quality:

Address: 0xE7E7E7E7E7 ✓ Auto Reconnect

Virtual Trajectory

Memo

```
---
SYS: Crazyflie 2.0 is up and running!
SYS: Build 0:47ee696b2d3c (2018.10)
MODIFIED
SYS: I am
0x3034363130374713002800 27 and I
have 1024KB of flash!
CFGBLK: v1, verification [OK]
DECK_DRIVERS: Found 16 drivers
DECK_INFO: Found 0 deck memory.
DECK_CORE: 0 deck enumerated
```

Clear | Task dump | Propeller test

Thrust (%) 0%

Status

Connected
Temperature C : 0.00C
Temperature F : 32.00C
Thrust : 0.00

Directional Information
Yaw : 0.00
Pitch : 0.00
Roll : 0.00

Adjust Thrust (%) Enter
Enable Motors

Trajectory

1) Set Points

Z : 0	Z : 0	Z : 0
X : 0	X : 0	X : 0
Y : 100	Y : 0	Y : 0

✓ Use Custom PID

2) UDP Settings

Port :

Vicon System - "255.255.255.255"

Connect UDP Stream

Progress (%) : 0%

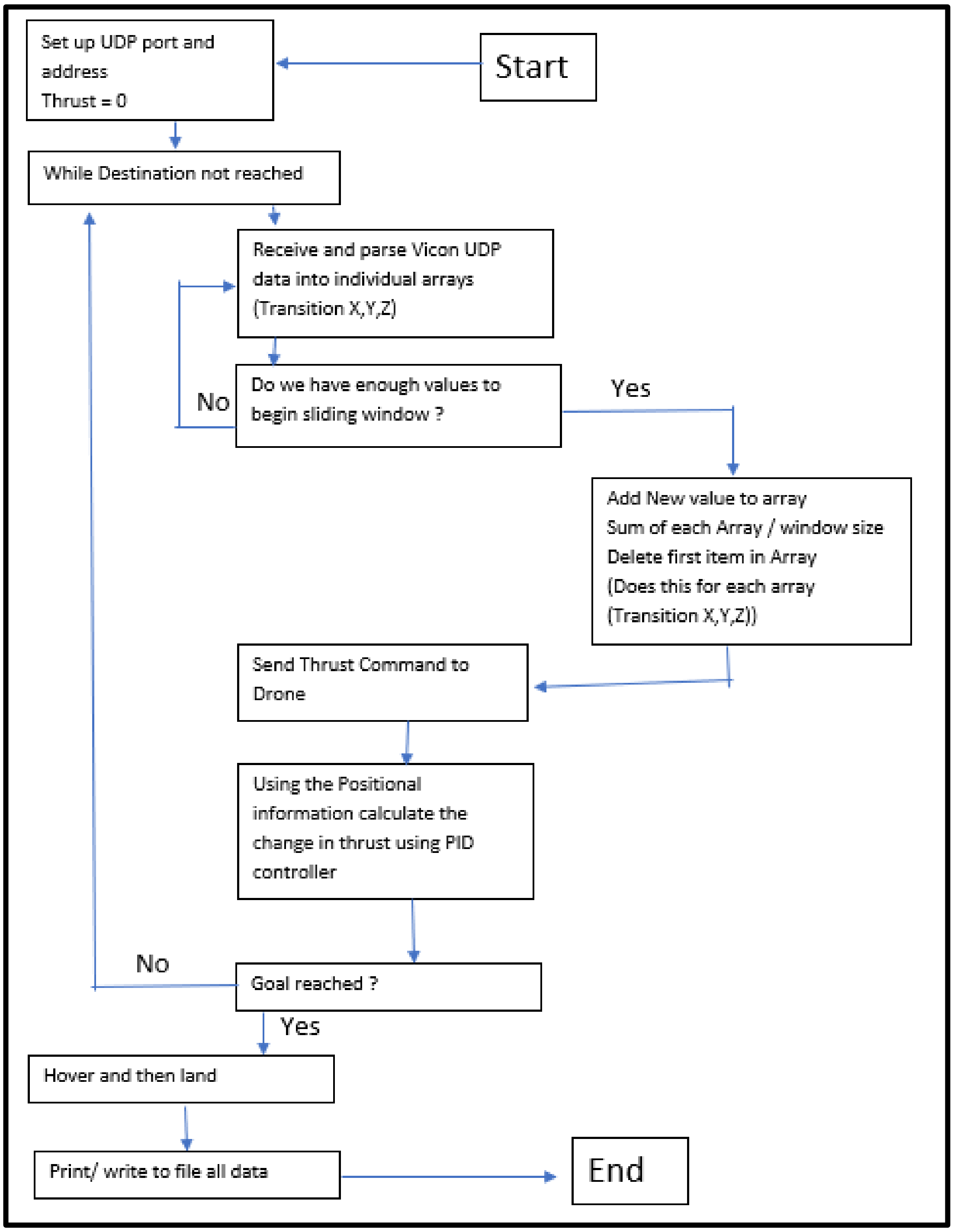
ZMQ Commands

Pitch :
Roll :
Yaw :
Thrust (%)
Enter

No input-device found, insert one to fly.

[Figure 60, request drone to ascend by 100mm in the Y axis, dead locked system]

Note: after the end of each lab session backups were made of the client code, for the purpose of the demonstration a back up will be used which only allows for single set point traversal.


Start
Set up UDP port and address
Thrust = 0
While Destination not reached
Receive and parse Vicon UDP data into individual arrays (Transition X,Y,Z)
Do we have enough values to begin sliding window ?
No
Yes
Add New value to array
Sum of each Array / window size
Delete first item in Array
(Does this for each array (Transition X,Y,Z))
Send Thrust Command to Drone
Using the Positional information calculate the change in thrust using PID controller
Goal reached ?
No
Yes
Hover and then land
Print/ write to file all data
End

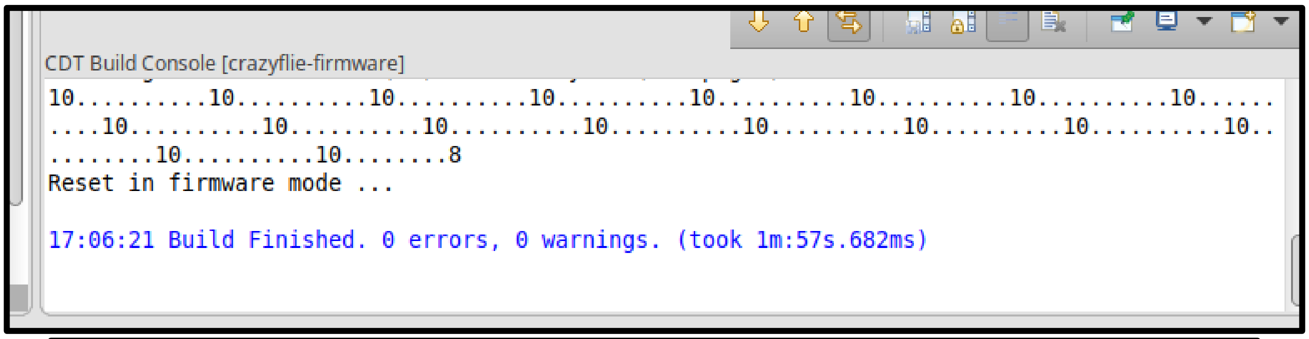
[Figure 61 ,ConnectUDP button flow of commands]

4.7. Testing

Testing for both the client and firmware was a time-consuming process which required many stages to reach one testable phase.

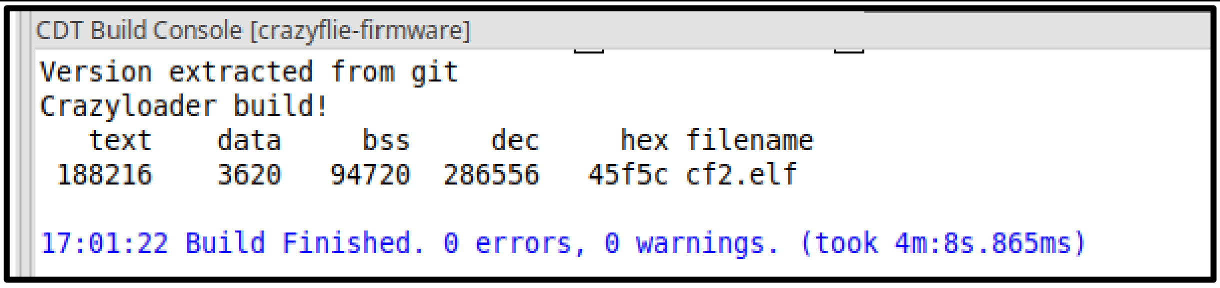
For example in order to fully test any functionality that would change parameters in the drones firmware be it thrust or yaw, the tester would first have to open the client, search for the drone and complete initial log requirements [7 seconds] and as frequently as 2/3 times the transmitter would fail to initially connect to the drone (this increases to approximately 5/6 when the drone has not been reset for 1 hour and is in a small room with many WIFI and Bluetooth enabled devices interfering with the transmitter). This all causes every test even for the smallest change to take up to 1 minute of just set up.

Similarly, for booting changes in the firmware back onto the drone via Save -> Clean -> Build. Each of these stages takes anywhere between 20 – 258 seconds each to complete. (This time can be cut significantly with the use of a SWD adapter or possibly by allocating more RAM to the VM)



CDT Build Console [crazyflie-firmware]

```
10.........10.........10.........10.........10.........10.........10.........10......
....10.........10.........10.........10.........10.........10.........10.........10..
.......10.........10.......8
Reset in firmware mode ...

17:06:21 Build Finished. 0 errors, 0 warnings. (took 1m:57s.682ms)
```



CDT Build Console [crazyflie-firmware]

```
Version extracted from git
Crazyloader build!
   text    data     bss     dec     hex filename
 188216    3620   94720  286556   45f5c cf2.elf

17:01:22 Build Finished. 0 errors, 0 warnings. (took 4m:8s.865ms)
```

[Figure 62,63 time taken to clean, build and reboot the CrazyFlie with new firmware]

4.7.1.1. Power issues

One unfortunate set back when testing was the lifespan of the battery in the CrazyFlie, with only 7 minutes of flight time. This made days set out specifically for testing PID tuning and other firmware alterations particularly difficult and drawn out.

Flight time with stock battery: 7 minutes
Charging time with stock battery: 40 minutes

After measuring the battery used in this project, 3827 volts increased to 3832 in 90 second. (5-volt increase) therefore, 1-volt charge per 18 seconds. On a normal test of the PID (40-60 seconds) the drone would consume an estimated 58 volts.

58 volts X 18 seconds = 1044/ 60 = 17 minutes

For every 1 minute of flight time, the drone will need to charge for 17 minutes (At this stage of the project it would be relevant to note the battery may have worn down or somehow been damaged after countless experiments, these values serve only as indication of flight time available in any one +-7-hour lab session)

When testing any code that will make use of the CrazyFlie 2.0 it is of utmost importance that the tester ensures the battery is charged over 3500 Volts. Any value lower than this will force the drone to enter a low power mode where packet data will lose priority and your code will cause erratic behaviour from the drone.

The drone should be thought of as a wired device that can enter a wireless mode temporarily.

4.7.1.2. PID Testing / Tuning

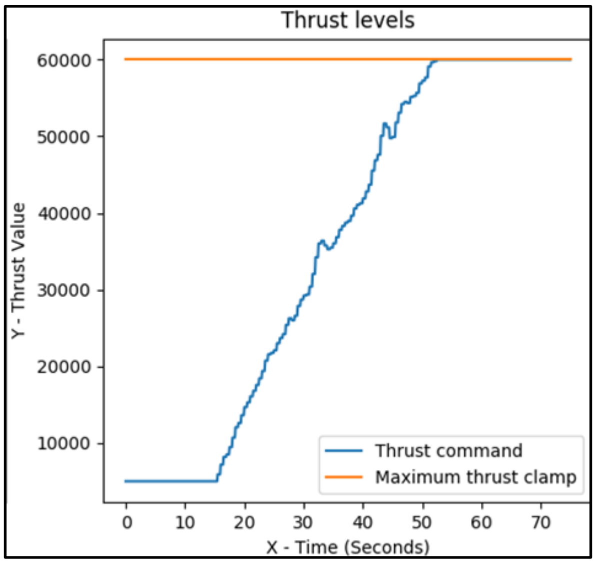
Client PID

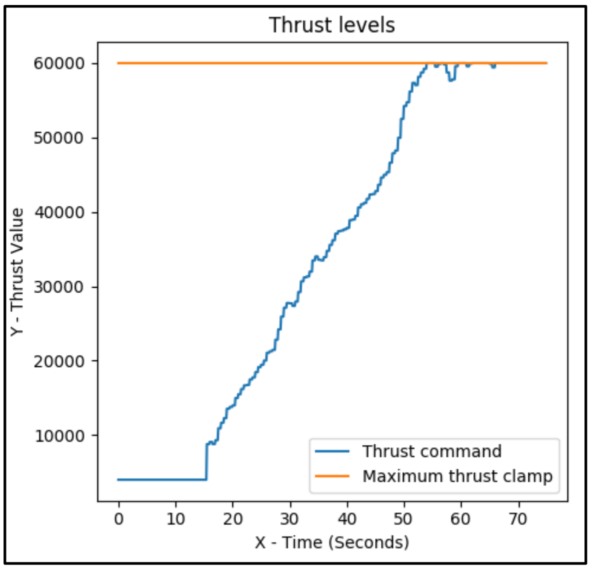
An optimal PID controller for any device requires a unique tuning. What is meant by this is that, each arm in the control loop must be adjusted to a specific value for optimal results to be achieved with regards to the propeller speed (torque).

One popular starting point is to set the value for **Kp** to something reasonable, dependant on the project and range of values (e.g. in this project upper thrust levels of 60000 are allowed so a relatively high **Kp** was selected) then continue to set each remaining arm to half of its predecessor in the order Kp -> Kd -> Ki.

	Kp (Proportional)	**Kd** (Derivative)	**Ki** (Integral)
Visualization	Kp	Kp x 0.5	Kd x 0.5
Values	10	5	2.5

Due to the significance Ki has on the entire calculation of torque, it was lowered to a relatively miniscule amount when compared to Kd and Kp. Kp and Kd shared the relationship of Kd = Kp / 2.





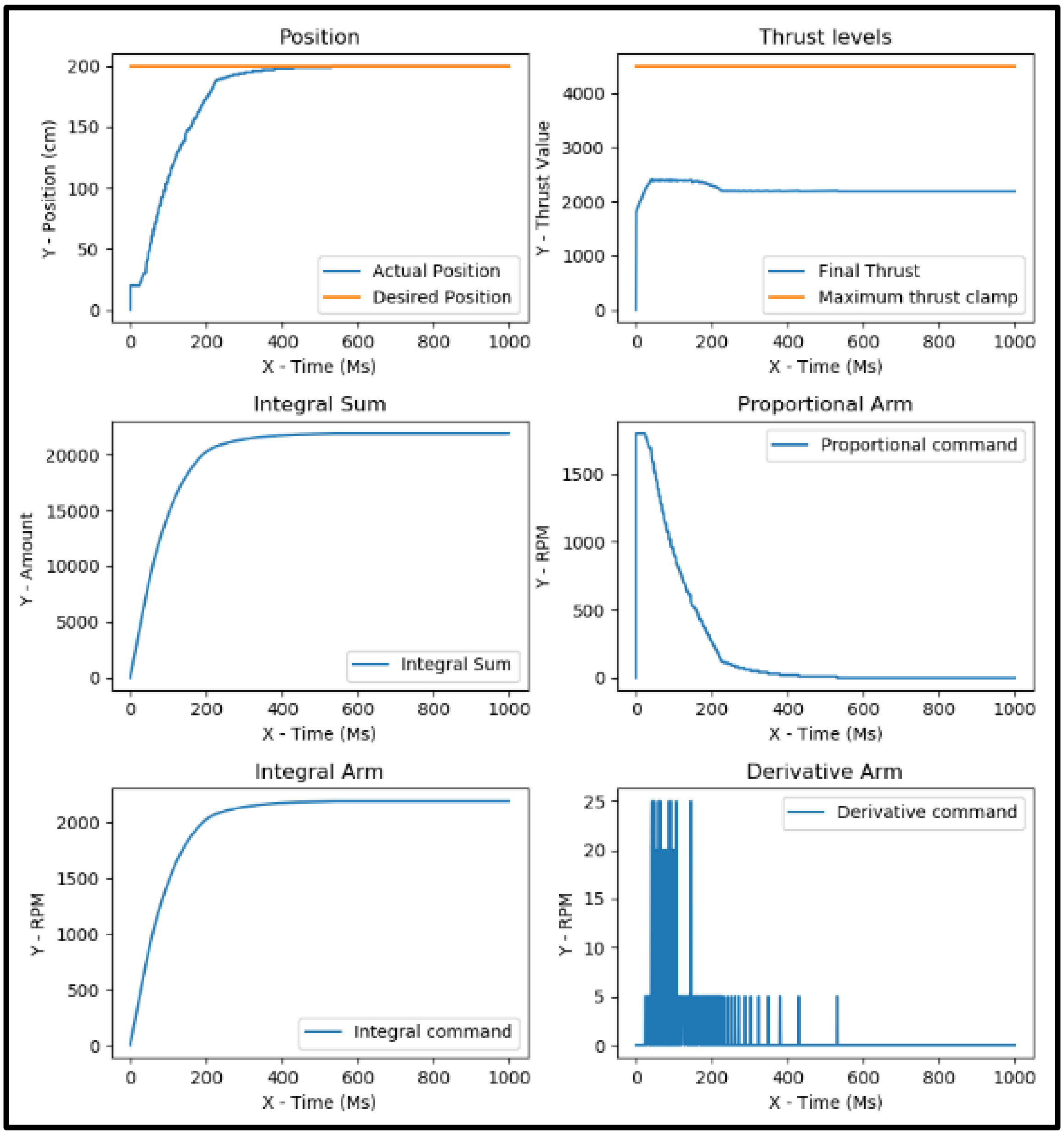
[Figure left 64, P = 25, I = 3, D= 15] **[Figure Right 65, P=30, I= 3, D=15]**

Figures above represent the PID controller running within the drone requesting maximum thrust, at second 57 in figure 51, the controller requested a lower thrust due to a change in measured position.

Note : Restart the drone after each test to reinitialize all firmware values otherwise gyroscopic readings may defer from actual orientation due to a bug within the current firmware version (yaw is constantly decreasing at a rate of +-0.01 each second) [Version 2018.10]

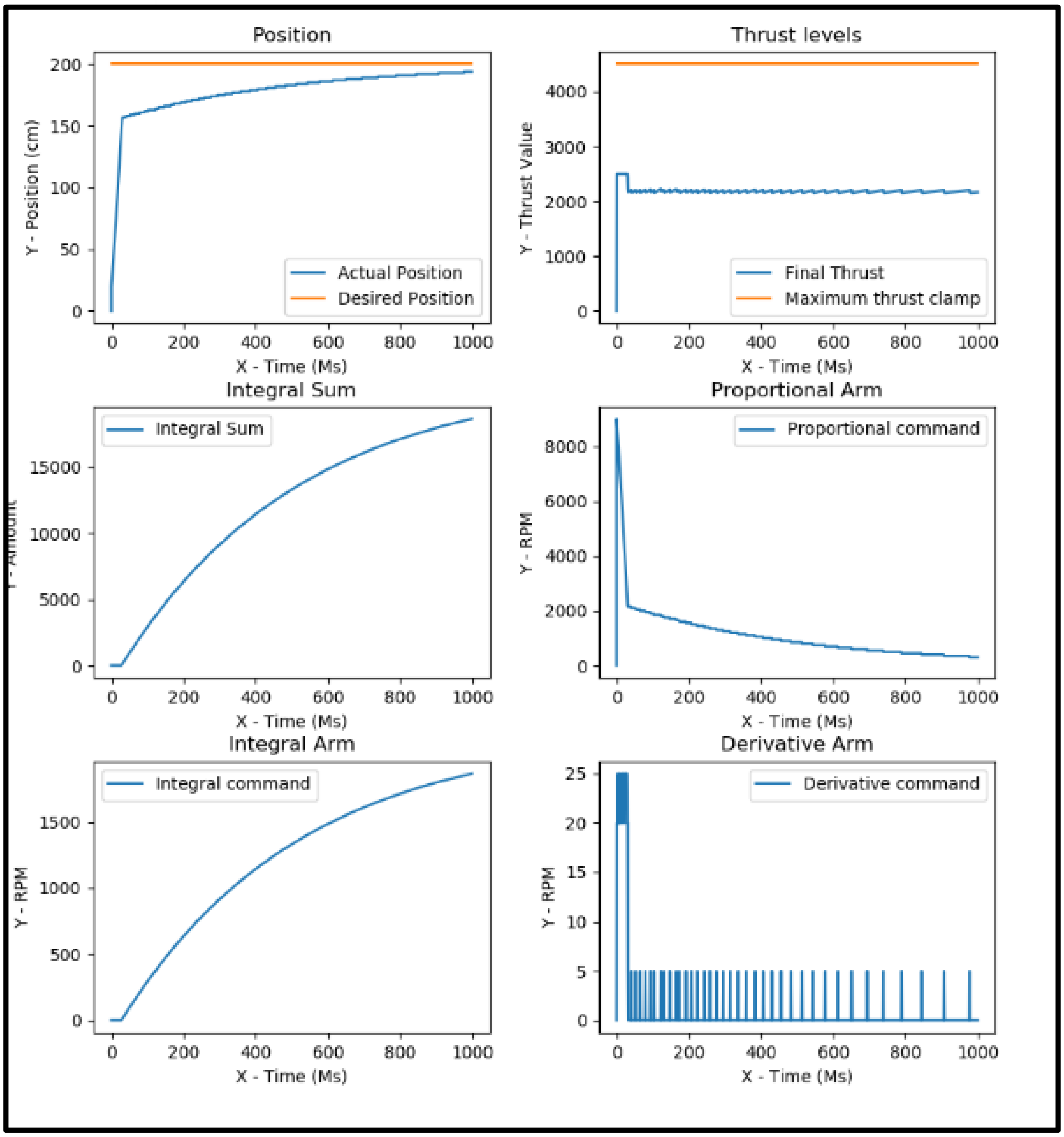
PID Simulation

Testing within the bounds of the simulation allowed for faster results with regard to the controller output and PID performance improvement. Values that are achieved within the simulation do not exactly mirror those recorded while operating the drone, but they do offer a generalized view of how the controller operates given certain dataset in a controlled environment. Below examines scenarios with vastly different tunings of PID values.



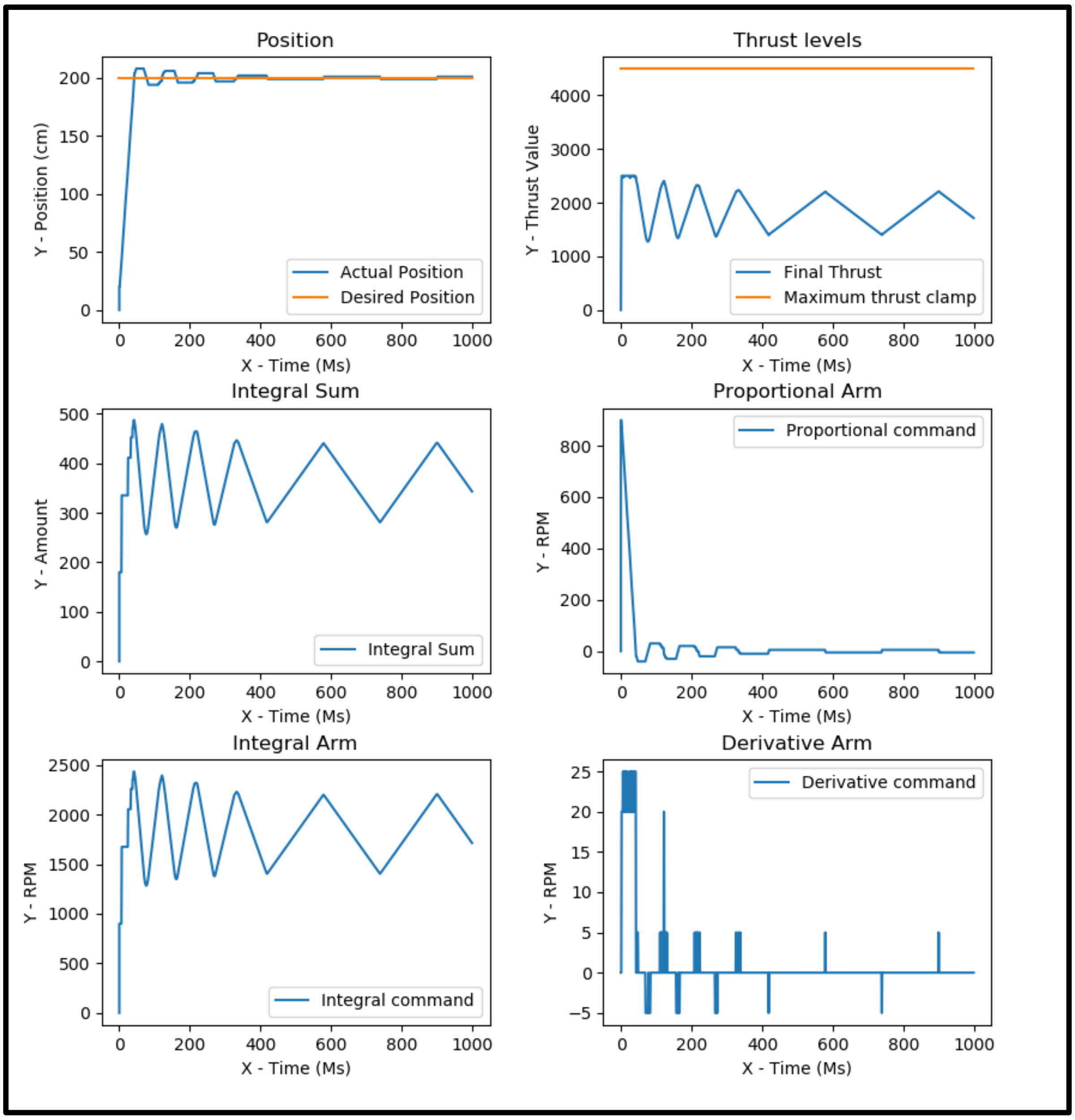
. **[Figure 66, Ki = 0.1,Kd = 5,Kp = 10]**

General tuning that has liberal integral arm growth as time increases, moderate Kp growth when error is large and a Kd arm that ensures faster growth with larger error and slower growth with smaller error.



[Figure 67, Ki = 0.1,Kd = 5,Kp = 50]

Faster initial ascent tuning that has a higher Kp value which drives the thrust high initially but causes a significant decrease in Ki sum growth due to the large error values being eradicated quickly leaving only smaller errors to accumulate in the integral sum which results in a slower arrival at the destination. Increasing Ki would be beneficial in this scenario.



[Figure 68, Ki = 5, Kd = 5,Kp = 5]

Unified tuning that sees all arms set to the same value. Due to the relatively large output of Ki, Kd is unable to slow the growth of the other arms causing an over shoot. The zig zag shape of the integral arm is due to the negative error terms entering the sum.

4.7.1.3. UDP Testing

UDP data stream values were validated against each other through a different stand-alone client. The purpose of this was to bypass the initialization of the CrazyFlie client. Accurate measurements including noise were initially captured and examined, then algorithms were fine-tuned until a stable averaged result could be produced for each field (e.g. Transition X).

4.7.1.4. Safety harness

An innovative piece of technology often goes hand in hand with a hefty price tag and unfortunately, so is the case with the CrazyFlie 2.0 drone, at the time of writing a new drone on Seeeds website currently costs $180. (Seeed, 2019)

So, to minimize damage to the drone, 2 safety products have been acquired and created to keep the drone flying for longer as it will not sustain as much damage from falls and collisions.

The first is a CrazyFlie 1.0 frame used to protect the motors that has now been modified to attach to the CrazyFlie 2.0s larger body. This frame is multi-purpose as it also holds the mo-cap markers in place.

The second is a rig that was used when experimenting at home throughout the lifecycle of this project. It was once a simple CD rack and now serves as a stable platform which limits the movement of the CrazyFlie drone to only the Y values (Altitude, Up/Down). It makes use of two lines of fishing wire threaded through the drone to restrict movement, it is also padded to absorb a large amount of the energy resulting from the drone free falling.

4.7.1.5. Faulty Equipment

The most time-consuming error to fix that was faced within the project was masquerading as a threading error within Python. The error thrown was flagging a file location at /LibUSB but as it was described as a threading error the initial thoughts were that too many processes must be running at the same time (receiving data stream, parsing, sliding window, PID loop, communication with the CrazyFlie) so much of the code was refactored until the client resumed normal operation (2 weeks of alterations with some results proving fruitful and others not but with little explanation as to why they were not working correctly other than “…Threading.py”). Eventually research was conducted on the driver used within the radio transmitter and the problem was solved by using another transmitter with a different driver installed.

4.8. Project Management

4.8.1. Task List

#	Task Name	Description	Duration (weeks)
1	Initial Report	Write the initial report covering research and plan for the project's duration.	3
2	Design interface layout prototypes	Design the GUI that will be used to control the drone.	1
3	Design interface to receive coordinates from user	Further the design and implement functionality to the interface that will allow the user to interact with the Quadcopter using specific coordinates.	5
4	Develop PID controller	Using the given specifications of the Quadcopter develop a PID controller algorithm that will ensure said drone can maintain flight following a specific trajectory or in a hovering position.	3
4.1	Develop Proportional Arm	Create the code necessary for the proportional arms functions	1
4.2	Develop Integral Arm	Create the code necessary for the integral arms functions including a clamp	1
4.3	Develop Derivative Arm	Create the code necessary for the proportional arms functions	1
4	Optimise PID Controller	As faults will be found initially with the controller, this tasks goal is to optimise and fix faults.	2
5	Achieve manual flight in 1 dimension Using client	To control the quadcopter in a safe software environment, the Crazyflie PC client will be used.	2
6	Achieve flight using given trajectory	By implementing all features of the PID controller and making use of the CrazyFlie PC client the drone shall attempt to follow a given trajectory in 1 dimension supported by the Vicon vison system.	3
6.1	Implement hover function	Control the quadcopter to hover at varying heights.	2
6.2	Implement directional movement	Control the quadcopter to move in varying directions	2
7	Final Report	Continuously update the final report document with new information regarding my findings and progress.	7
8	Testing	Ensure that all erroneous behaviour from the Quadcopters operation using the custom PID and GUI is resolved.	2
9	Testing (Quadcopter Operation)	Locate any unusual exceptions regarding the quadcopters operation in flight and resolve them.	1

10	Testing (GUI Operation)	Examine the usability of the GUI while in the hands of a user and attempt to improve upon short falls.	1
11	Research Python coding language	To modify the Crazyflie PC client, knowledge of Python is needed.	3
12	Integrate use of Global vision system	This shall ensure that the system is able to view the location of the quadcopter.	1
13	Research operational use of Vicon systems	To track quadcopter in flight.	1
14	Conduct initial experiments with Crazyflie	To familiarize oneself with the hardware used in this project	1
15	Additional features	Further on the project by implementing features out of the scope.	2

[figure 69 , task list]

4.8.2. Time Plan

Initial

		University Calendar weeks										
#	Task Name	4	5	6	7	8	9	10	11	12	13	14
1	Topic overview research	■	■									
2	Conduct initial experiments and research Crazyflie device		■	■								
3	Research operational use of Vicon systems			■								
4	Initial Report		■	■	■							
5	Achieve manual flight (Using client)				■							
6	Research Python coding language				■	■						
7	Develop PID controller						■	■	■			
8	Optimise PID Controller									■	■	
9	Further Testing of PID algorithm											■

		University Calendar weeks																
#	Task Name	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
10	Design interface layout prototypes.	■																
	Exam Period		■	■														

11	Design interface to receive coordinates from user				■	■												
12	Integrate use of Global vision system						■	■										
13	Fully implement hover function								■	■								
14	Fully Implement directional movement										■	■						
15	Achieve flight using given trajectory												■	■				
16	Testing (GUI Operation)														■			
17	Testing (Quadcopter Operation)														■			
18	Final Report								■	■	■	■	■	■	■			
19	Additional features															■	■	■

Updated

		University Calendar weeks										
#	Task Name	4	5	6	7	8	9	10	11	12	13	14
1	Topic overview research	■	■									
2	Conduct initial experiments and research Crazyflie device		■	■								
3	Research operational use of Vicon systems			■								
4	Initial Report		■	■	■							
5	Achieve manual flight (Using client)				■							
6	Research Python coding language				■	■						
7	Develop PID controller						■	■	🟥	🟥	🟥	🟥
8	Optimise PID Controller									🟥	🟥	🟥

		University Calendar weeks																
#	Task Name	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
9	Design interface layout prototypes.	■																
	Exam Period		■	■														
10	Integrate use of Global vision system				■													
11	Customize CrazyFlie client					■												
11	Catch up on previous PID tasks				■	■	■	■	■									
12	Further Testing of PID algorithm									■	■							
13	Further designed interface to receive coordinates from user										■	■						
14	Fully Implement single dimensional											■	■					
15	Achieve flight using given trajectory												■	■	■			
16	Testing (GUI Operation)														■			
17	Testing (Quadcopter Operation)														■			
18	Final Report								■	■	■	■	■	■	■			
19	Additional Time															■	■	■

[figure 70, Time plan]

4.8.3. Risk Analysis

Risk	Severity (L/M/H)	Likelihood (L/M/H)	Significance (Severity x Likelihood)	How to avoid	How to recover
Power failure	H	H	HH	Alert user of low battery	Charge Quadcopter.
Motion capture marker damage	M	M	MM	Keep markers clean at all time and avoid handling directly with hands to ensure they stay clean and functional	Place markers in a container with warm soapy water to clean them
Damage to Vicon tracking system	H	L	HL	Always follow all rules outlined by staff in charge with the wellbeing of the system	Seek advice from the ICT department
Motor failure	H	L	HL	Ensure CrazyFlie has passed its self-check when it powers on (Indicated by green LED when initially turned on)	Replace / Repair motor.
Bugs within firmware	H	M	HM	Keep firmware up to date	Revert to previous stable builds
Loss of Signal	H	H	HH	Alert user once signal begins to drop below a required minimum.	Decrease the distance between the Client and the Quadcopter.
Damage to quadcopter	H	M	HM	Only use device in an open area where minimal damage can occur from quadcopter dropping or encountering obstacles.	Replace / Repair the required component.
Bugs within CFLIB	H	M	HM	Keep CFLIB up to date	Consult the wiki or forum for information on how to recover from specific bugs
Injury	L	M	LM	Ensure to always work in a safe environment while using protective gear	Allow enough time for injury to heal.
Loss of program data	H	L	HL	Make regular back ups	Recover from one of the backups made.

Accidental update of crucial software used by Virtual Machine causing inoperability of programs (Bitcraze V18.01)	H	H	HH	Ensure to always make use of back-ups and ensure that you fully understand where a link points to before running it.	Recover previous state from saved back up.
Damage to transmitter	H	M	HM	When traveling back and forth ensure to keep the transmitter in a waterproof case that will minimize movement	Another transmitter will have to be bought and will take 2 weeks to be delivered.
Unforeseen circumstances	H	H	HH	Make use of good programming practices and familiarity of system to prevent any unpredictable errors occurring while in use of the system.	This is task dependant, use available resources to find a solution.

[Figure 71, risk analysis]

5. Critical Evaluation

5.1. Definition of done

According to the description of this project, the system is only complete if the following goal is satisfied according to the definition of working.

The goal and definition of working in verbatim are:

- "The goal of this project is to develop a PID controller for the Crazyflie flying robots supported by a global vision system."
- "The work is to develop and optimise a PID controller for the drone and develop a graphic interface to interact with the drone so that the drone can fly and follow a given trajectory by the user."

To that end we can verify that the project is done once we can answer yes to all the following questions:

1. Has a PID controller been researched, developed, tuned and optimised?
2. Is the PID controller supported by a global vison system?
3. Has a graphical user interface been created to allow users to interact with the drone?
4. Can the drone fly to a specific set point?
5. Can the drone fly to a sequence of different setpoints (a trajectory / a path)?

5.2. Technical Achievements

- Development of a means of intercepting and parsing a UDP stream using network sockets.
- Creation of client within an existing system to utilize crucial features and hardware.
- Proficiency of a previously unknown language obtained. (Python)
- Ability to write code to connect with multiple external devices using alternative methods. (UDP for Vicon, TCP for Crazyflie 2.0)
- Ability to identify errors in complex systems.
- Successful completion of extensive research is visible in practical deliverables across various fields including Networking (TCP/UDP), Embedded systems (Creation of custom drivers used in the Crazyflie 2.0 firmware), Machine Vision (Vicon tracking system), Multiple programming language proficiency (Python, C/C++)
- Implementation of a complex programming algorithm (Client PID controller)
- Proven all theoretical aspects initially discussed of PID controllers in a practical setting.
- Proven ability to work well with embedded systems. (various sensors and devices used in this project)
- Single directional autonomous movement
- The drone can travel to a single desired setpoint with a certain level of accuracy on the Y axis while secured within a restrictive rig.

5.3. Review of Achievements vs project objectives

By comparing the two sections above (Definition of Done and Technical achievements), questions 1-4 of the D.O.D can be answered with yes unequivocally. However, whether 3-dimensional trajectory was specified in the description or not, 3-dimensional autonomous flight is an objective that I did set out for myself to aim for and at the end of this project life cycle I am reluctant to stop and disappointed that I was not able to achieve it.

What has been achieved:

- Vicon Tracker objects created, configured, tracked, faults identified, reconfigured and finally tracked again with evident understanding of how each sub section functions.
- PID controller researched, created, tuned, refactored and tuned again with evident understanding of the model in both a theoretical and practical setting with a simulation PID (Theoretical) as well as a client PID for the Crazyflie (Practical) created for multisystemic use.

- Creation of a client to interface with the drone and command it to travel to a single setpoint (with 4 possible methods explored, elaborated on and fully functional code implemented for 3)

5.4. Are the goals met according to time plan?

Tasks 1 – 6 were completed according to plan without any major obstacles, task 7-9 were significantly understated when designing the time plan and were eventually completed 4 weeks after the initial time expected due to dependencies on the Vicon tracking DataStream and its positional information to be able to close the loop of the PID controller to verify its functionality.

To make up for lost time the time plan was adjusted accordingly, leaving less time for additional features and more time spent on core functionality.

All other tasks and initially unplanned deliverables (e.g. creation of PID simulation, multiple methods explored to send commands to drone) were completed according to the time up until task 14.

5.5. Future Development Work

- Utilize the already developed PID controller to regulate pitch, yaw and roll to achieve 3-dimensional autonomous trajectory. The work left to do to achieve this goal includes making use of the already parsed x, y and z rotational data along with the logging framework to receive the current orientation data recorded by the gyroscope to adjust according to the direction the drone must travel.
- As long as the error I had experience within my drone's firmware which had an ever-decreasing yaw value persists, stable 3D flight will not be possible. I used multiple firmware versions and still experienced this error on both my client and the Bitcraze client. This is an issue that will/ would have been fixed by the Bitcraze team.
- Use of the UDP data transfer method was time consuming to implement and provided minimal results, exploring the use of the Vicon SDK may provide more accurate methods to cancel out noise and receive more accurate positional results.
- 2 bugs are currently deadlocking the system, updating CFLIB to its latest version fixes the API related BUG.
- Extend the functionality of the client to support an ecosystem to control a swarm of CrazyFlie's
- Given more time an aesthetically appealing addition to the client would be a method to toggle between each of the cameras used by the Vicon system, this unfortunately is unobtainable through use of the current UDP stream (Possibly made easier using VRPN and a simple call method in the Vicon SDK).
- A bicycle / quadcopter hybrid with 4 large lightweight propellers on the outward facing sides of the 2 wheels, these two wheels would somehow split apart and fold up into the same quadcopter shape as the CrazyFlie and fold back down into a bicycle.

5.6. PROBLEMS FACED, AND LESSONS LEARNT

This project was extremely difficult to complete up to the point that I have completed it.

Each separate segment of this project although interesting, brought with it new and vastly different problems. These problems all required working knowledge of the specific part of the system the problem arose from and when all parts were used together, it required specific knowledge of all different parts of the systems.

From recalibrating an object within the Vicon system as it was no longer being detected due to a worn-down marker, *Axis Swapping*, malfunctioning equipment, limited amounts of time with hardware necessary to complete this project, deadlocking caused by the API or even spending over an hour finding a client breaking bug caused by the misspelling of "Received" in PyCharm (A python hacker I was not in week 6). Throughout this project I was always painfully aware of the next problem to rear its head around the proverbial corner and here I document some of the major problems faced.

This Section documents common major issues faced and includes notes on how to recover in the hopes that others do not spend countless hours searching for answers to no avail.

5.6.1. VIRTUAL MACHINE:

As is the case with most open sourced software, the Virtual Machine supplied by Bitcraze for its Crazyflie 2.0 drone has its fair share of flaws. The majority of the files within the machine are riddled with outdated documentation on how to fix certain program breaking errors.

When running a fresh BitcrazeVM version 18.01 build using VirtualBox, few things are immediately available. Many libraries and programs need to be downloaded for essential programs to work such as many of the CrazyFlie PC Clients dependencies, this is the application that will be used to control the drone. The inability to access this fundamental program on the Virtual machine and finding a solution took up a large amount of time as there was very little documentation online regarding this system and how to fix any problems that occur.

Time that was initially reserved for PID development tasks couldn't be completed without the use of the client. Having spent over 30 hours initially installing, reinstalling, updating and uninstalling libraries and programs was the biggest problem I faced in the early stages of this project.

Lesson Learnt:

I now feel I am fully aware of how volatile systems of this type (Virtual / Open source) are and understand that errors are present. To combat this, I have made saving the state of the VM in a separate file from the one I am currently working on a number 1 priority as to not lose countless hours of work at 3 hour / milestone intervals (whichever comes first).

- Make use of all resource's available online and more specifically on GIT HUB
- Make frequent use of back-ups to keep loss of data at a minimum.
- Ask for guidance from your assigned tutor when an impasse is reached so that development can continue from a different angle.

Notes:

- Bitcraze Vm Version 2018.12 is a stable and fully functional build of the machine needed to develop and perform all tasks outlined in the *"Aims and Objectives"* section. This build has been updated to XUbuntu 18.04 which makes the installation of development programs like PyCharm much easier with a simple "sudo snap install pycharm-community –classic " command.
- Use version 18.04 if any future updates lead to issues within the Virtual Machine similar to those described in 18.01.

5.6.2. CrazyFlie Client

The most notable problem with developing alongside an existing program that I faced was familiarization of the structure and data already present. As it was already a standalone program there are features it must complete to continue working as it should and issues that I assumed were to do with the code I was writing (Threading.py) were eventually found out to be a piece of hardware malfunctioning. Had I written down the name of the driver for the Crazyflie PA that I installed I would have been able to save at least a week of time to improve other aspects of the system.

The deadlocking experienced in the final week of the project was something I was unable to recover from, and although it meant the system was only able to travel to a single set point by using a backup version of the client I feel the situation is a reflection on my technical scope of the project as a whole. As with all of the other errors in the system and their causes, they have all been identified. There was no stone left unlabelled within the client code.

Lesson Learnt:

When using another system as a foundation for your own, you must be completely familiar with all components otherwise things such as method, API or hardware specific errors will become much harder for you as a programmer to identify and fix as you lack the understanding to identify them.

Open source projects are not going to be perfect, and their documentation is going to be even less so. Critical research skills to be able to trace similar issues back to one root cause are what I learnt from the client and although I was unable to fix the final bug, I was able to conclude and identify it along with many others along the way through critical research.

5.6.3. PID development

Transforming a complex computational model into a fully-fledged algorithm was something I had prior experience with so although research on the control theorem was a long process, implementation of the algorithm was not as time consuming as the other parts of the system. I feel this is due the amount of time spent working through the model until I fully understood it.

Lesson Learnt:

More time spent researching and working through the theory of a model in the early stages results in less time needed to implement said model in the later stages.

5.6.4. Vicon Tracker System

Axis swapping / Flipping is simply put, the needle in the haystack of this project which led to incorrect orientational transition values being sent to the client and a problem everyone who would have progressed to this stage would have experienced, whether knowing about it and finding a way around it or being oblivious to it. Although I am aware of the rig obscuring 1 or more of the markers leading to this issue, this issue was still common with configurations mark 1-4 even when not attached to the rig as there must be axis defining markers.

Lesson Learnt:

With the use of multiple systems, it is easy to get overwhelmed and end up focusing on 1 specific task or error at a time. The issues this creates are errors that may be mislabelled as being associated with the current task you are working with as it is what you are altering. This is not the case and every other aspect of the system must be thought of as just as liable, regarding cause of errors, as the one you are currently working with.

5.6.5. Research

As evident in the first aim within the Aims and Objectives subsection, there is a considerable amount of knowledge needed for this project. This knowledge cannot be obtained by simply searching through multiple sources and summarizing what you have found, the only way to obtain this knowledge is to attempt and to fail. As one of my lectures once told me, “the person who has gone and found code online and copy and pasted it into their IDE and gotten it working has learnt nothing, only those who have tried and failed can understand how a system operates ”.

Lesson Learnt:

Research must go hand in hand with a practical context, I understand how PID controllers work not because I’ve spent hours on end reading online resources, but because I’ve taken that knowledge summarized its principles and then applied them and failed multiple times until I finally understood how the model works and which parts are needed to perform specific functions.

5.6.6. Overall Lessons Learnt Conclusion

Throughout the development of the simulation, client, firmware and physical devices used to secure and track the drone many things have been learnt. It started with the use of Linux for the first time and writing my first python “hello world” program and ended with the implementation of a complex regulatory model updating values in an embedded device while making use of various sensors to translate positional estimates into a change in force which was sent to the quadcopter to enable autonomous flight. Within 1500+ lines of coded simulations for the PID and essential UDP networking practices, and 4 software systems of which were eventually encapsulated and refactored into the 1 CrazyFlie pc client demonstrated, I can say to a degree of certainty that many lessons were learnt through the development of this project.

Most notably:

- The importance of time management and adequate requirements scoping in the early stages of the project life cycle.
- Valuable in-depth research techniques into theory and practicality of the PID computational model along with numerous other background information on related topics and systems which would be used in tandem with the controller.
- I have learnt that preparing myself for obstacles that WILL eventually arise in systems that may seem impassable from the current view point, will always have a solution after stepping back, analysing each possible scenario and investigating accordingly.
- Whether it is creating a program to solve a puzzle or to make a drone fly, logical thinking can only ever improve after being tested past your limits to force you to adapt to the learning curve and better your standard of thinking. After 700+ hours spent on this project I feel that I am now ready for any challenges that may come my way in future embedded software development cases.

6. Conclusion

The systems produced by Bitcraze although may have their faults on the developmental side once only remnants of their original code remain, is logistically sound and without blame par 1 malfunctioning transmitter and 2 bugs causing a deadlock in the dying weeks of the project .Without their attention to provide answers to questions within the Bitcraze forum at a moment's notice, the wiki and the fully documented API none of the methods discussed would have been possible. As stated, Method 3 which uses the API to send *Big 4* values to the drone was the method chosen for its dependability and accuracy when dealing with packets. Although it was the method chosen for this project, method 4 may prove to be easier to implement and streamlined for this project context as evident in the code supplied in the program (with no more than external positional parsing, averaging and then sending the data to the drone needed). 1 dimensional autonomous flight was achieved and the foundations for 3-Dimensional flight have undoubtable been laid for anyone who views this report and supporting deliverables.

Initially I assumed that connection to the Vicon system would be as easy as connecting a plug and play Usb webcam into a computer with no thoughts that a UDP network and sliding window methods would need to be researched and set up. This was undoubtably my biggest miscalculation when allocating the time plan. Research into the methodology and setting up clients and servers to test operability took an unprecedented 2 weeks + 3 weeks on decoding and implementing averaging methods on the data. These positional estimates riddled with noise captured on the Vicon system that had a deviation of over 15% was not a simple task to solve. After taking the initiative to change the techniques used twice I feel the sliding window technique solves the problem adequately however there are better methods that would most likely have been provided with the Vicon SDK. In this case, a more complicated set up (Vicon SDK) may have provided a faster and more useful end product.

As evident by the simulation results, a PID controller has its benefits but when implementing complex systems, the choice of PID instead of PI is one that should be well thought out and supported by the need of the altering of rate of change brought upon by the derivative arm rather than a clamp once the integral sum is driving enough gain. Clamping within the integral arm proves to be just as useful and although complex without an understanding of the data, it is very effective.

Flight once thought of as an unattainable feat of nature, but now through great advancements in engineering, pioneering intellectuals have spent countless hours researching and implementing systems that each build upon the next to allow us to reach this goal. Soon multiuse regulator systems controlling autonomous drones could be in the binary hands of neural networks or even an advancement in technology that hasn't yet been discovered.

Breaking further and further out of my disciplines confined computer science template and merging into aspects of engineering and arithmetic has been an enjoyable and pivotal experience. My only hope is that eventually in years to come, this report may prove useful to at least 1 other who would continue this work and develop further into the unknown.

"We choose to go to the Moon in this decade and do the other things, not because they are easy, but because they are hard; because that goal will serve to organize and measure the best of our energies and skills, because that challenge is one that we are willing to accept, one we are unwilling to postpone, and one we intend to win, and the others, too." – John.f.Kennedy (Kennedy, 1962)

REFERENCES

Akesson, K., 2017. *Lecture.* [Online] Available at: https://www.youtube.com/watch?v=Q_mG2gbplRE [Accessed 13 10 2018].

arnaud, 2019. *Bitcraze ZMQ.* [Online] Available at: https://wiki.bitcraze.io/doc:crazyflie:client:pycfclient:zmq [Accessed 10 01 2019].

Bitcraze, 2018. *CrazyFlie 2.* [Online] Available at: https://www.bitcraze.io/crazyflie-2/ [Accessed 10 11 2018].

Bitcraze, 2018. *The Crazyflie Python API.* [Online] Available at: https://wiki.bitcraze.io/doc:crazyflie:api:python:index [Accessed 29 09 2018].

Bitcraze, 2019. *cfc Client.* [Online] Available at: https://wiki.bitcraze.io/doc:crazyflie:client:pycfclient:index#configuration_block [Accessed 10 02 2019].

Bitcraze, 2019. *crazyflie.* [Online] Available at: https://www.bitcraze.io/crazyflie/ [Accessed 08 10 2018].

Bitcraze, 2019. *CrazyRadio Pa.* [Online] Available at: https://wiki.bitcraze.io/projects:crazyradiopa:index [Accessed 05 10 2018].

Bitcraze, 2019. *Logging and Parameter frameworks.* [Online] Available at: https://wiki.bitcraze.io/doc:crazyflie:dev:arch:logparam?s[]=toc [Accessed 01 01 2019].

Bodenham, D., 2019. *Adaptive Filtering and Change Detection for Streaming Data.,* London: Imperial College.

Crouch, T. D., 2018. *Sir George Cayley Biography.* [Online] Available at: https://www.britannica.com/biography/Sir-George-Cayley

Douglas, B., 2018. *MatLab Tech Talks.* [Sound Recording].

Eurotherm, 2018. *Principles of PID Control and Tuning.* [Online] Available at: https://www.eurotherm.com/temperature-control/principles-of-pid-control-and-tuning [Accessed 21 12 2018].

Fred, 2017. *Eclipse foundation.* [Online] Available at: https://www.youtube.com/watch?v=WV5KcVy8vEs

Gibiansky, A., 2019. *Quadcopter Dynamics, Simulation, and Control,* s.l.: s.n.

Goodman, J., 2005. *Monte Carlo Methods,* NYU: Online.

Graham C. Goodwin, S. F. G. M. E. S., 2018. *National Instruments.* [Online] Available at: http://www.ni.com/white-paper/3782/en/#toc2 [Accessed 15 02 2019].

Grewal, M. S., 2001. *Kalman Filering Theory and practice.* Second ed. Canada: Wiley-Interscience .

Icady, 2009. *Code project.* [Online] Available at: https://www.codeproject.com/Articles/36459/PID-process-control-a-Cruise-Control-example

JetBrains, 2019. *Pycharm.* [Online] Available at: https://www.jetbrains.com/pycharm/ [Accessed 05 12 2018].

Kennedy, J. F., 1962. *PRESIDENT JOHN KENNEDY'S RICE STADIUM MOON SPEECH.* Houston, Texas: s.n.

Limited, V. M. S., 2018. *Vicon Tracker User Guide,* Oxford: s.n.

Maliszewski, S., 2015. *Next Stage Pro.* [Online] Available at: http://www.nextstagepro.com/retroreflective-markers.html

Pri, A., 2018. *Yaw,Roll and Pitch defined.* [Online] Available at: https://www.quadcopterflyers.com/2015/02/quadcopters-yaw-roll-and-pitch-defined.html

QT, 2019. *QT Designer Manual.* [Online] Available at: https://doc.qt.io/qt-5/qtdesigner-manual.html [Accessed 10 01 2019].

Rabault, J., 2019. [Online] Available at: https://folk.uio.no/jeanra/Microelectronics/PIDController.html

Radcliffe, T., 2018. *Python Vs. C/C++ In Embedded Systems.* [Online] Available at: https://www.activestate.com/blog/python-vs-cc-embedded-systems/

Seeed, 2019. *Seeed.* [Online] Available at: https://www.seeedstudio.com//Crazyflie-2.0---p-2103.html [Accessed 16 03 2019].

Spero, J., 2019. *Finantial Times.* [Online] Available at: https://www.ft.com/content/cdaa19e6-0f97-11e9-a3aa-118c761d2745

Stauffer, Y., 2004. s.l.: cireseerx.

Urquizo, A., 2019. *WikiPedia.com.* [Online] Available at: https://en.wikipedia.org/wiki/PID_controller [Accessed 26 10 2018].

Vicon, 2018. */www.vicon.com.* [Online] Available at: https://www.vicon.com/motion-capture/engineering [Accessed 30 12 2018].

VirtualBox, 2019. *VirtualBox wiki.* [Online] Available at: https://www.virtualbox.org/wiki/Downloads [Accessed 04 01 2019].

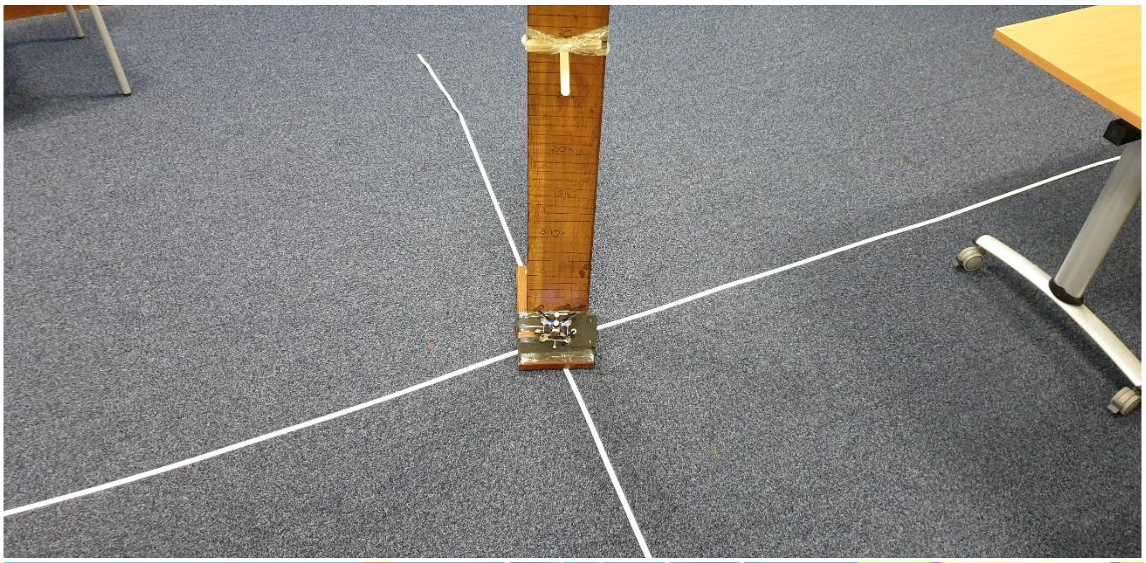
Wikidot, 2019. *zeromq.* [Online] Available at: http://zeromq.org/ [Accessed 05 10 2018].

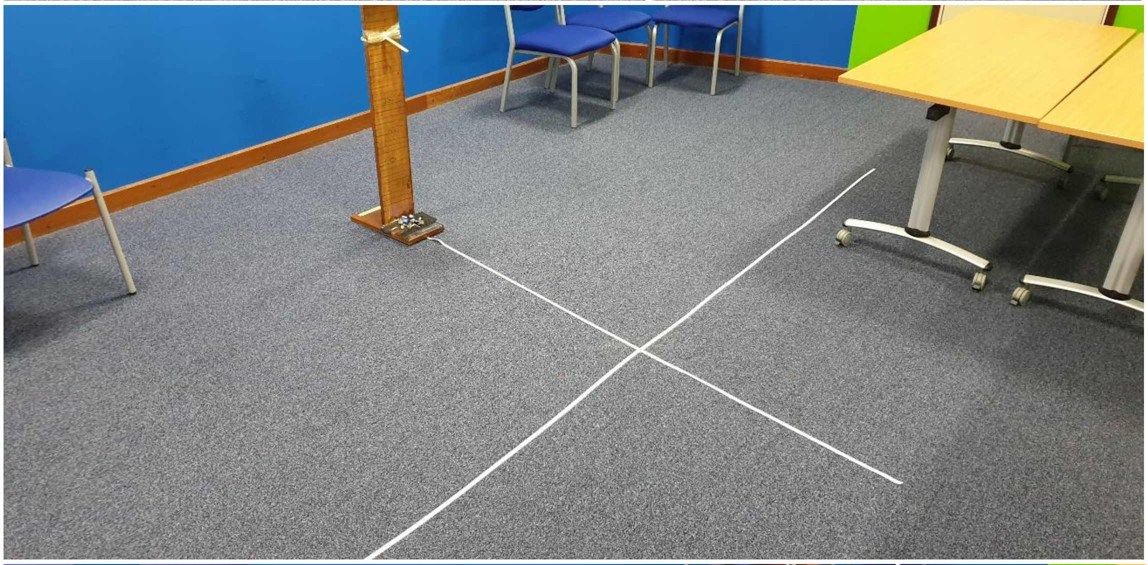
Wodehouse, C., 2018. *C vs C++.* [Online] Available at: https://www.upwork.com/hiring/development/c-vs-c-plus-plus/

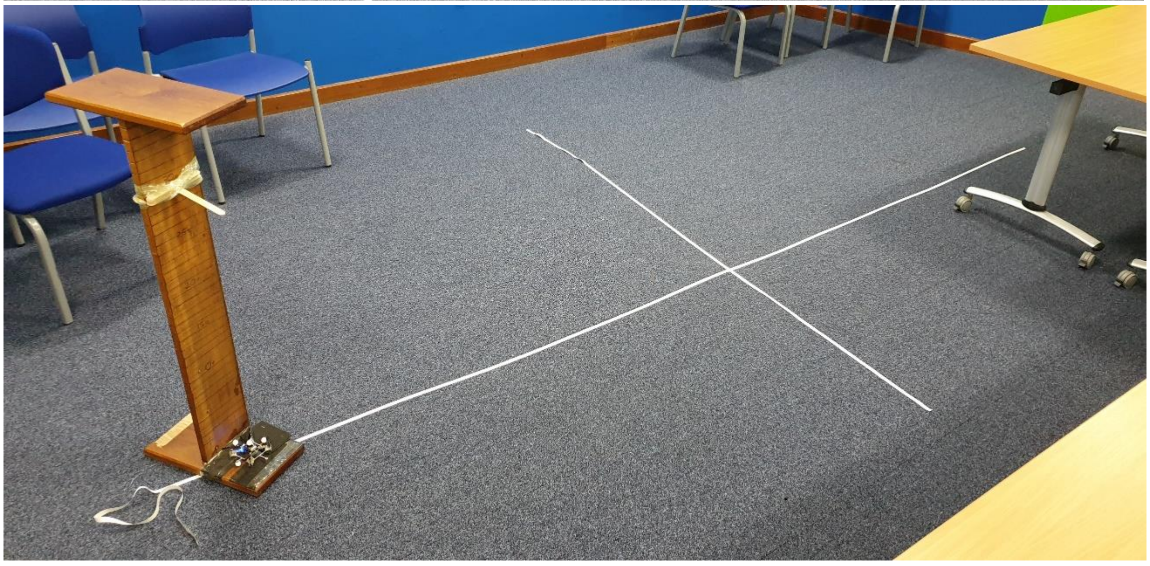
Libraries used in development:

7. Appendix

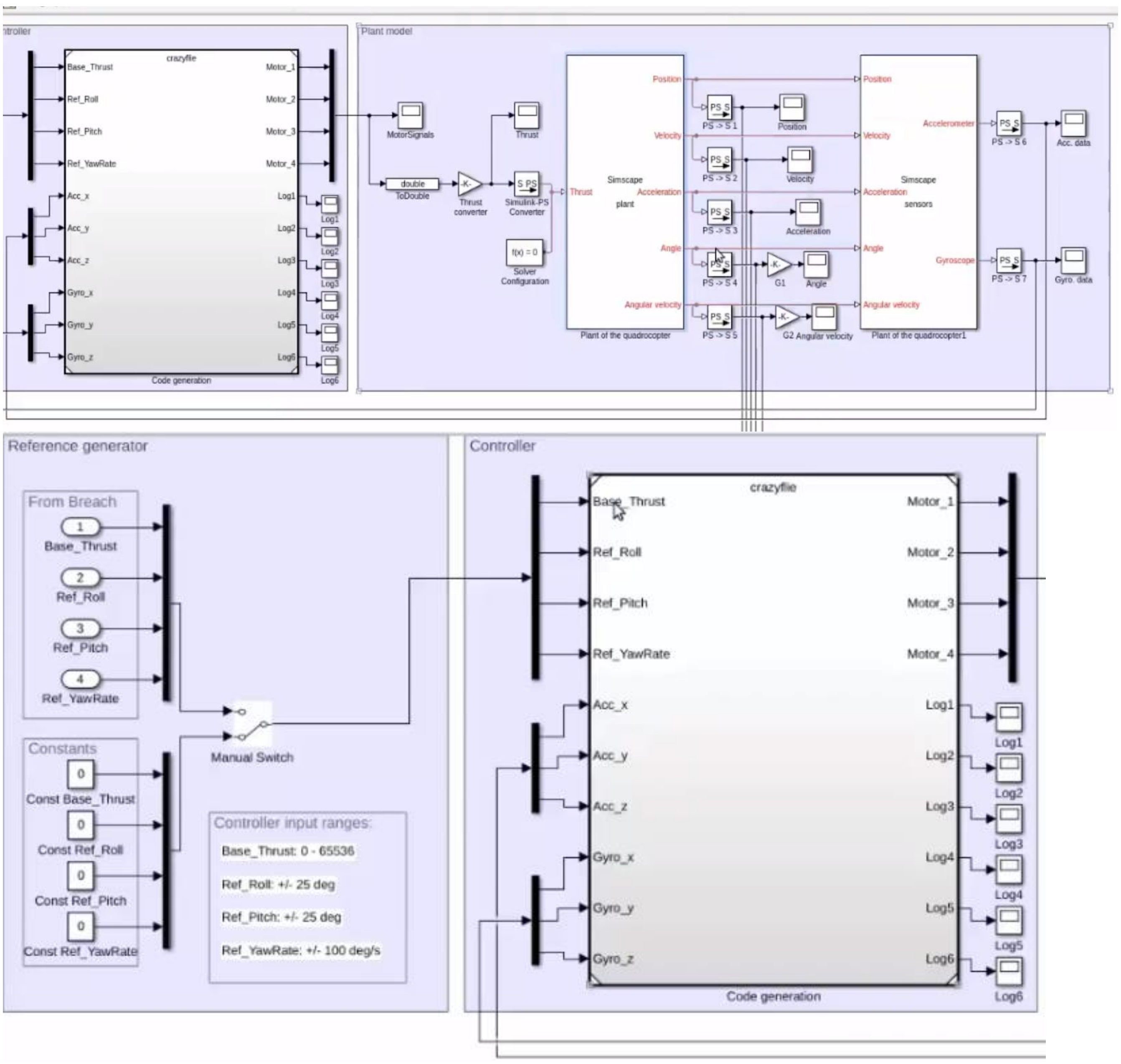
7.1. Appendix A: Experiment conducted that discovered Axis Swapping / flipping







7.2. Appendix B: Supporting images



Controller input ranges:

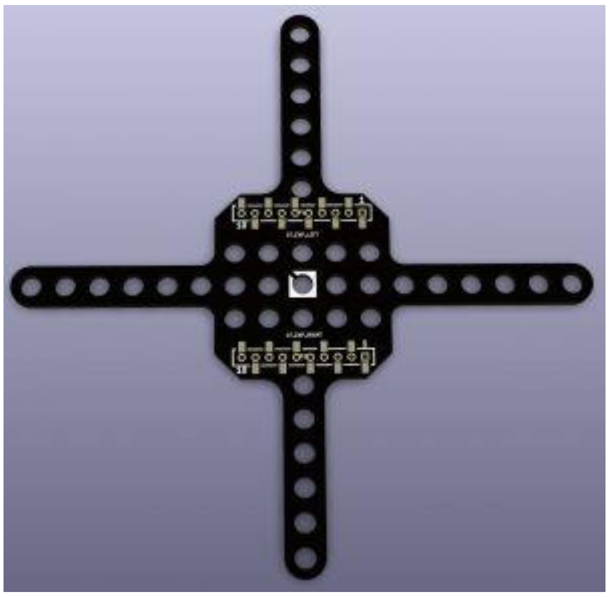
Base_Thrust: 0 - 65536

Ref_Roll: +/- 25 deg

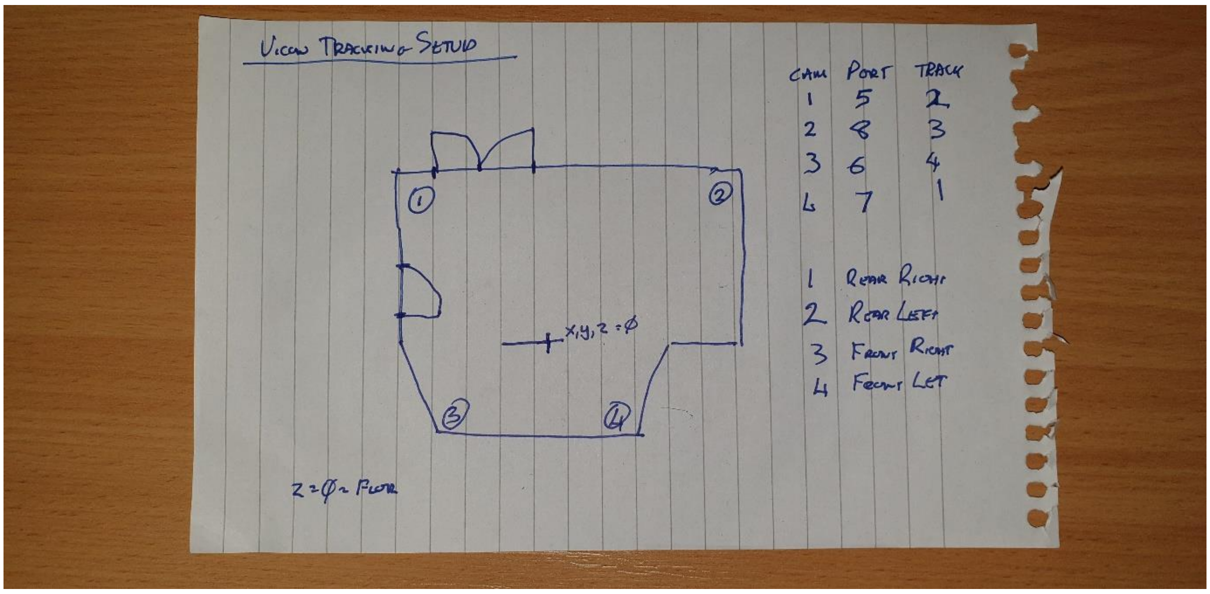
Ref_Pitch: +/- 25 deg

Ref_YawRate: +/- 100 deg/s

[Figure 72 above, CrazyFlie controller diagram (Akesson, 2017)]



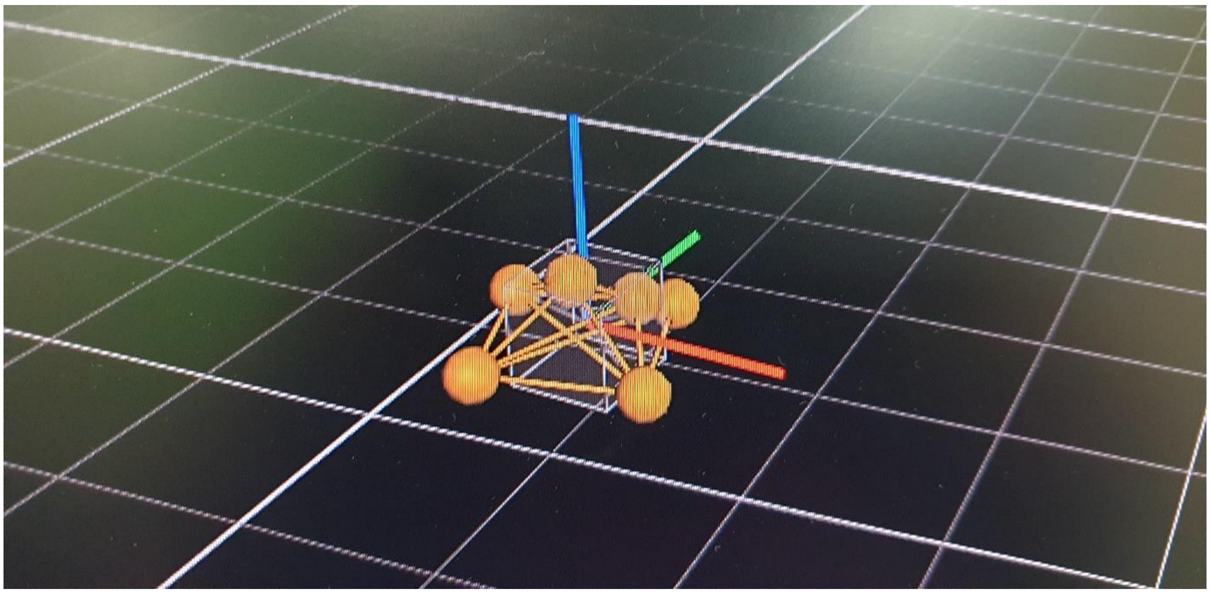
[figure 73, Mocap deck for the CrazyFlie 2.0 which would have provided easily configurable layouts for the markers but could not be found online to purchase]



CAM	PORT	TRACK
1	5	2
2	8	3
3	6	4
4	7	1

1	Rear Right
2	Rear Left
3	Front Right
4	Front Let

[figure74 , Layout of the green room at Hull university where the Vicon equipment was located]



[figure 75 , 6 marker drone in Vicon environment]

```
## Folder description:
```
./ | Root, contains the Makefile
 + init | Contains the main.c
 + config | Configuration files
 + drivers | Hardware driver layer
 | + src | Drivers source code
 | + interface | Drivers header files. Interface to the HAL layer
 + hal | Hardware abstaction layer
 | + src | HAL source code
 | + interface | HAL header files. Interface with the other parts of the program
 + modules | Firmware operating code and headers
 | + src | Firmware tasks source code and main.c
 | + interface | Operating headers. Configure the firmware environement
 + utils | Utils code. Implement utility block like the console.
 | + src | Utils source code
 | + interface | Utils header files. Interface with the other parts of the program
 + platform | Platform specific files. Not really used yet
 + tools | Misc. scripts for LD, OpenOCD, make, version control, ...
 | | *** The two following folders contains the unmodified files ***
 + lib | Libraries
 | + FreeRTOS | Source FreeRTOS folder. Cleaned up from the useless files
 | + STM32... | Library folders of the ST STM32 peripheral libs
 | + CMSIS | Core abstraction layer
```
```

Figure above, file structure of CrazyFlie firmware

7.3. Appendix C: Installations

Pycharm: Download from https://www.jetbrains.com/help/pycharm/install-and-set-up-pycharm.html

1. sudo snap install pycharm-community --classic

Swig: Download from http://www.swig.org/

1. `cd' to the directory containing the package's source code and type `./configure'
2. `make' to compile the package.
3. 'Sudo make install'

cMake: Downloaded from www.cmake.org

1. sudo apt install cmake
2. cmake /home/bitcraze/Downloads/vrpn-master/

Vrpn: Download from https://github.com/vrpn/vrpn

- Requires Swig
- Requires CMake
- Compile the main library (ie.: root of VRPN) and the quat library.
- in Quat file uncomment Linux_64
    - Make MakeFile
- In /home/bitcraze/Downloads/vrpn-master/python_vrpn/
    - make vrpn-python

- Hand-written Python classes that work with Python 2.7 and 3.2.
    - Compile the main library (ie.: root of VRPN) and the quat library. Then, go to the "python" folder. Before making the binary, you have to define the "PYTHON_VERSION" environment variable to the version of python you want to compile it for (ie.: "3.2", "2.7" ...). And you have to put the resulting vrpn.so shared library (found in $HW_OS/$PYTHON_VERSION) in the python module folder or in a path defined in the PYTHONPATH environment variable.
- The "essai_*.py" are simple python files that show you how to use this module (the single difference is a call to python3.2 or python2.7 interpreter).
    - cd vrpn
    - tar xzvf python_vrpn.tar.gz
    - cd python_vrpn
    - edit Makefile.python and set the environment according to your setup
    - make vrpn-python
    - (as root) make install-vrpn-python
- ISSUES :
- - if vrpn_Tracker fails to load on symbol handle_update_tracker, comment out line 1124 of vrpn_Tracker.h (//static int VRPN_CALLBACK handle_update_rate_request (void *, vrpn_HANDLERPARAM);). I didn't find other fix so far.
- http://act.usc.edu/publications/Preiss_ICRA2017.pdf

7.4. Appendix D: Useful information

You also need to understand what the LEDs mean.

- Power on and all is good, but sensors are not yet calibrated: The blue LEDs (2 and 3) are fully lit and the front right LED (1) is blinking red with 2 seconds interval. Put the Crazyflie 2.0 on a level surface and keep it absolutely still to calibrate.
- Power on and all is good: The blue LEDs (2 and 3) are fully lit and the front right LED (1) is blinking red twice every second.
- Radio connected: The front left LED (4) is flickering in red and/or green.
- Battery low: The front right LED (1) is fully lit in red. It's time to land and re-charge the battery.
- Charging: The back left blue LED (3) is blinking while the right back blue LED (4) is lit.
- Boot loader mode: The blue LEDs (2 and 3) at the back are blinking approximately once every second.
- Self-test fail: The right front LED (1) is repeatedly blinking five short red pulses with a longer pause between groups. (Bitcraze, 2019)

VRPN

- A comprehensive guide to walk any developers through the use and implementation of VRPN servers/ clients can be found at: http://www.vrgeeks.org/vrpn/tutorial---use-vrpn
- and a description of all the files and folders can be found at (I strongly recommend any one using VRPN to read this entire page before commencing any further) http://www.cs.unc.edu/~taylorr/cyberinfrastructure/vrpn_html/vrpn_standard_stuff.html

Default config file

```
{
 "writable" : {
   "input_device": "radio://0/100/2M",
   "link_uri": "",
   "flightmode": "Normal",
   "open_tabs": "Flight Control",
   "trim_pitch": 0.0,
   "slew_limit": 45,
   "slew_rate": 30,
   "trim_roll": 0.0,
   "max_thrust": 80,
   "min_thrust": 25,
   "max_yaw": 200,
   "max_rp": 30,
   "client_side_xmode": false,
   "auto_reconnect": false,
   "device_config_mapping": {},
   "enable_debug_driver": false,
   "input_device_blacklist": "(VirtualBox|VMware)",
   "ui_update_period": 100,
   "enable_zmq_input": false
 },
 "read-only" : {
   "normal_slew_limit": 45,
   "normal_slew_rate": 30,
   "normal_max_thrust": 80,
   "normal_min_thrust": 25,
   "normal_max_yaw": 200,
   "normal_max_rp": 30,
   "default_cf_channel": 10,
   "default_cf_speed": 0,
   "default_cf_trim": 0
 }
}
```