Formation Control of Crazyflies

Bryce Mack Chris Noe Trevor Rice

November 2017

Advisors: Dr. Ahn & Dr. Wang

Abstract

This project studies the implementation of a distributed formation control algorithm using several nano quadcopters called Crazyflies. This will be accomplished using the Robot Operating System (ROS) for programming and a Kinect 360 module for localization. The Kinect module will track each quadcopter and provide localization information to our controller. From there, we will implement a distributed control algorithm to a swarm of Crazyflies. This will be done with the main control law running on remote computer and transmitting control signals to Crazyflies through a Crazyradio dongle. The radio has the ability to send multiple radio control signals to several Crazyflies at once, which allows us to individually control the swarm. Furthermore, the algorithm contains various PD and PID controllers for quick response and stabilization. With these controls the system will achieve hovering and formation control within a specified workspace. Our control zone will be setup in a clean, confined, indoor space.

Contents

1 Introduction

Recent years have seen a significant progress in the technology development of unmanned aerial vehicles (UAVs). Particularly, the usage of multiple UAVs has received a lot of attention in both industry and military. For instance, a group of small UAVs could be used in a search and rescue mission in a harmful region for human.

In general, the individual UAV in a group has its own sensors and actuators. To fully coordinate the motion of multiple UAVs for certain common tasks, the fundamental questions become how to deal with sensing/communication among individual UAVs and how to design simple yet efficient local control strategies for each UAV. There exist extensive distributed cooperative control algorithms for general linear dynamical systems. Some results may be applicable to the distributed control of multiple UAVs. However few results are available in terms of experimental implementation of algorithms in real UAVs especially in the presence of various uncertainties including system dynamics uncertainty, and sensing and communication uncertainties. The core objective of this project is to design practically implementable distributed control algorithms for UAVs and to implement them using an agile nano quadcopter called Crazyflie.

2 Problem Statement

In this project, we will be using the Crazyflie 2.0 produced by Bitcraze (Fig. 1). The distributed control design will be based on the system model abstracted from Crazyflie. The sensing/communication and control strategies will be implemented and tested using a number of Crazyflies.

We chose to use this nano quadcopter over the others on the market because of its high quality components. This copter is also open source, so we will be able to program and control it using our own software. This copter is small for us to safely test indoors. It has 5-10 minutes of flight time with less than an hour charge time. Bitcraze also produces its own Crazyradio which can be used to control multiple Crazyflies at once. This was a big part of our overall project and with the Crazyflies we would not need to do much hardware development. All the hardware integration has already been taken care of by Bitcraze. However, we will be using the Robotics Operating System (ROS) to build and run any software on the bot. Figure 1 shows an image of a Crazyflie 2.0. Its specifications are given below.

- Weighs 27 g
- Size (WxDxH): 92x92x29mm (motor-to-motor and including motor mount

Figure 1: Crazyflie 2.0

feet)

- 20 dBm radio amplifier tested to more than 1 km range LOS with Crazyradio PA
- STM32F405 main application MCU (Cortex-M4, 168MHz, 192kb SRAM, 1Mb flash)
- nRF51822 radio and power management MCU (Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash)
- IMU: 3-axis gyro, accelerometer, and magnetometer
- Max recommended payload weight: 15 g

There are several main research problems to be addressed in this project:

1. Identify an approximate mathematical model to describe the dynamics of the Crazyflie.
2. Design a software simulation platform (Matlab/Simulink or ROS-based) which can be used to simulate various control designs.
3. Design control algorithms for configuration stabilizing control and trajectory tracking control of Crazyflies.
4. Develop an algorithm to estimate the yaw, roll, and pitch angles of Crazyflies using its onboard IMU.
5. Develop an algorithm to estimate (x,y,z) coordinates of Crazyflies using images captured by Kinect vision sensors.
6. Develop a distributed control strategy for formation motion of three Crazyflies.
7. Implement and test the robustness of control algorithms.

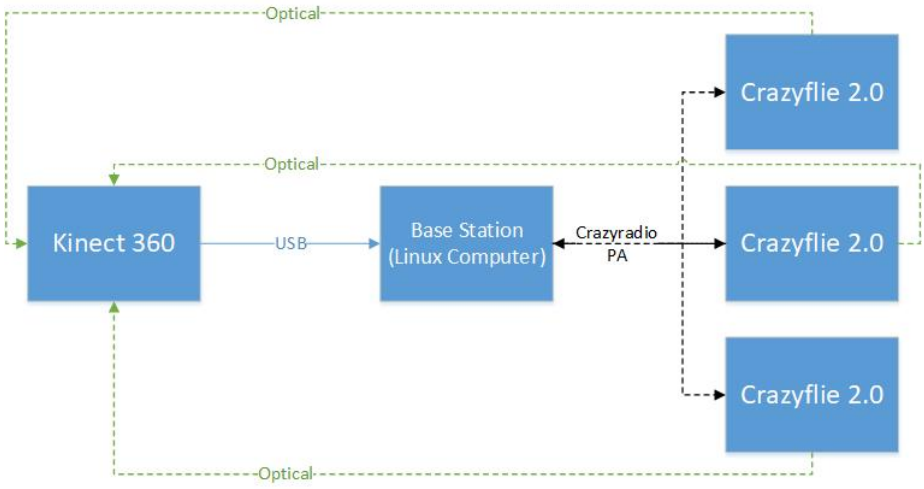

Figure 2: High Level Diagram of the System

3 Functional Requirements

While the main objective of the project is to design and implement distributed control algorithms for Crazyflies, the overall functional requirement of the project is focused on completing two primary tasks. The primary tasks are to make the Crazyflie hover, and then to implement a distributed control algorithm for a swarm of Crazyflies.

We will program and control the Crazyflies using a Crazyradio PA. This device is a USB radio dongle that can be plugged into any computer with a USB port. Our code base will be written in C/C++ and compiled in the Ubuntu Trusty 14.04 Linux environment using ROS. The control system will be run on the Crazyflies on-board chips, which will be doing most of the processing necessary to keep the bots stable.

In addition to reading the Kinect 360 localization data, the laptop will be sending localization information through the Crazyradio to update the control algorithm running on the Crazyflies. In the case of multiple Crazyflies, we will control them with one Crazyradio PA. We will be using GIT to track any documents or code that we develop for simulations, models, and demonstration.

Our first model to make the Crazyflie hover in one position will be based on the result in [1]. We will be modeling in Simulink and compare our simulations to those found in the paper. After this, we will begin experimenting with implementing the same algorithms in C/C++ and testing with the Crazyflie.

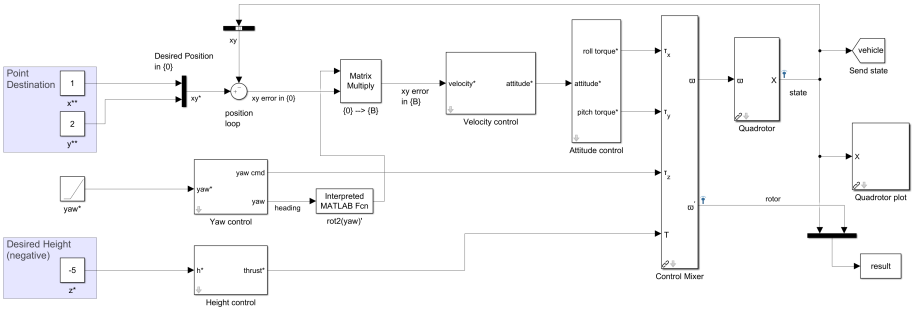

Figure 3: Simulink Model From Toolbox

3.1 High Level System Design

The High Level System Diagram is shown in Figure 2. The Kinect 360 modules communicate with the Base Station via USB. The Base Station takes the localization information from the Kinects. The localization information is then transmitted to the Crazyflies through the Crazyradio PA. The Crazyflies use the localization information to update the control algorithm running on their on-board chips.

4 Research Problem 1: Modeling and Control of Crazyflie

The quadrotor model included in the toolbox[2] is extremely detailed with how it is set up. There are dozens of parameters that are specific to the quadrotor that Corke and his research team based their model on.

We read multiple research documents on system identification for the Crazyflie. However, there were still parameters that were not covered in the research. The researchers focused on identifying thrust coefficients and rotor speeds. So, instead of doing our own system identification for the Crazyflie, we chose to leave the quadrotor model as is.

The model shown in Figure 3 is used to direct the quadrotor to a specific point. The model contains 4 PD controllers: Velocity, height, yaw and attitude.

4.1 Quadrotor Dynamics

This quadrotor model requires a transformation from the Inertial Frame to the Crazyflie Body Frame. The body frame that Corke used has the standard x-y axes but the positive z-axis is in the downward direction. This way, gravity corresponds to the positive z direction. While using this frame of reference for the Crazyflie, we must remember that all values entered for z will be negative, i.e., anything above the ground has a negative z value.

While researching quadrotors, [3] presented a simple, rigid-body model of a quadrotor, assuming low speeds. The equations defining the model are given by

$$\begin{aligned}
\ddot{x} &= (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)\frac{U_1}{m} \\
\ddot{\phi} &= \dot{\theta}\dot{\psi}(\frac{I_y - I_z}{I_x}) - \frac{J_R}{I_x}\dot{\theta}\Omega_R + \frac{L}{I_x}U_2 \\
\ddot{y} &= (\cos\phi \sin\theta \sin\psi + \sin\phi \cos\psi)\frac{U_1}{m} \\
\ddot{\theta} &= \dot{\phi}\dot{\psi}(\frac{I_z - I_x}{I_y}) - \frac{J_R}{I_y}\dot{\phi}\Omega_R + \frac{L}{I_y}U_3 \\
\ddot{z} &= -g + (\cos\phi \cos\theta)\frac{U_1}{m} \\
\ddot{\psi} &= \dot{\phi}\dot{\theta}(\frac{I_x - I_y}{I_z}) + \frac{1}{I_z}U_4
\end{aligned} \tag{1}$$

As they say in [3], the desired position and orientation (pose) of a quadrotor is hovering. Therefore, we can reduce Eqn. 1 using small angle approximations. Let $U_1 = mg + \Delta U_1$. Under these approximations the equations become

$$\begin{aligned}
\ddot{x} &= g\theta \\
\ddot{\phi} &= \frac{L}{I_x}U_2 \\
\ddot{y} &= -g\phi \\
\ddot{\theta} &= \frac{L}{I_y}U_3 \\
\ddot{z} &= \frac{\Delta U_1}{m} \\
\ddot{\psi} &= \frac{1}{I_z}U_4
\end{aligned} \tag{2}$$

The four inputs U_1,U_2,U_3 and U_4 represent the inputs for roll, pitch, yaw and thrust. These are the equations that Corke's model is based on. The 4

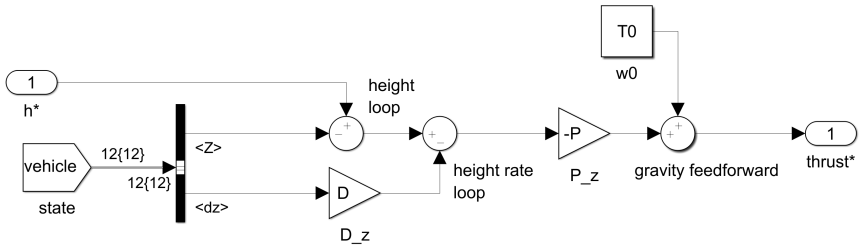

Figure 4: Original Height Controller

PD controllers are used to continuously update the input commands during simulation.

4.2 Hovering Control Design

Our first task with the Crazyflie is to have it hover at one point in 3D space. By following the experiments performed in [1], we will have the Crazyflie go from the ground to a specific point a few feet away but above the ground. The Crazyflie will hover stably in that position for up to 30 seconds. In addition to just hovering, we would be able to introduce disturbances to the bot and it would be able to overcome any disturbances to maintain stability.

In order for us to be able to test controlling a quadcopter, we needed a model that we could run to simulate inputs and disturbances. Instead of starting from scratch, we were able to use a model created by Peter Corke. We installed his MATLAB toolbox that has been developed by Corke and multiple graduate research students over the past two decades. Corke has also written a textbook [2] to explain how to use various parts of his toolbox.

Originally, the height controller used PD control and had a feed-forward thrust term that was used to compensate for the thrust needed to keep the quadrotor airborne.

Corke used this because his quadrotor weighed 4 kg and so it required a large amount of thrust to stay in the air. By including a constant feed-forward term, Corke eliminated the need for integral control. This allowed his quadrotor model to stabilize much faster since an integral controller reduces settling time.

Our Crazyflie only weighs 27 g and the thrust needed to keep it airborne is negligible. Dr. Wang advised us to introduce integral control into the height

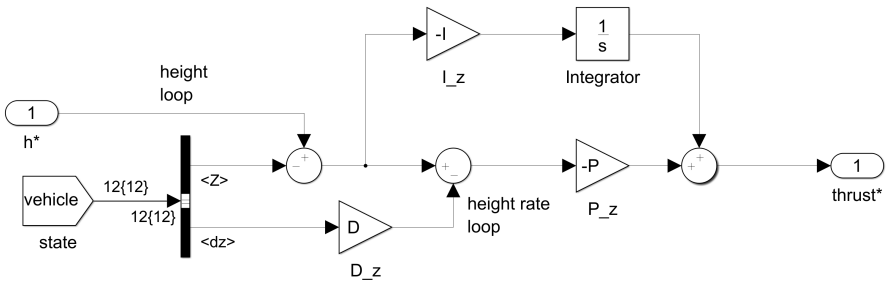

Figure 5: PID Height Controller

controller. The integral controller can be seen in Figure 5. This replaced the T0 term shown in Figure 4.

4.3 Formation Control Design

The baseline design for formation control will be based on the linearized quadrotor model, which can be represented as a double integrator given below

$$\ddot{x}_i = u_i, \;\; i = 1, \cdots, n$$

where x_i denotes the state variable and u_i is the control input. To illustrate our idea of design, in what follows, we simply assume $x \in \Re$ and $u \in \Re$. Further design Details will be presented in the final report.

The formation control is of the form

$$u_i = \sum_{j=1}^{n} a_{ij}(x_j - x_i) + \sum_{j=1}^{n} a_{ij}(\dot{x}_j - \dot{x}_i) \tag{3}$$

4.4 Simulation and Preliminary Results

After making the replacement in Figure 5, we needed to modify the control values to optimize overshoot and settling time. We modified the subsystem mask to include inputs for kd, kp and ki. These are the proportional, derivative and integral gains, respectively. Dr. Wang recommended that we use the control design of $kp * kd > ki$ with all control gains > 0. For the initial test, we chose to use

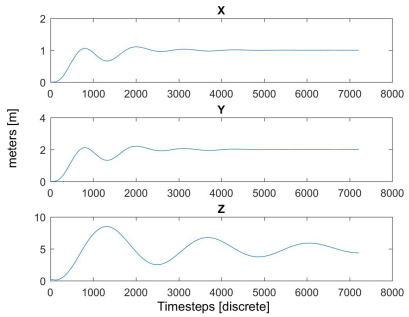

Figure 6: X,Y,Z Plots using control values in Eqn. 4

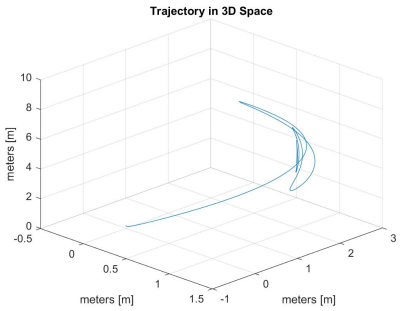

Figure 7: 3D Plot of Fig. 6

$$\begin{aligned} kp &= 5 \\ kd &= 5 \\ ki &= 10 \end{aligned} \tag{4}$$

For these simulations, we are focusing on the response of Z, shown in Fig. 6. By running the MATLAB command stepinfo, we got the step response information. With these control values, the overshoot was 70.2% and a settling time $> 30s$. We definitely want to reduce these and have $< 20\%$ overshoot and settling time $< 10s$.

After a few more trials, we learned how the controller responds to the control gains. Reducing kd decreased the overshoot. Increasing kp decreased the settling time. With this in mind we chose to use the following control values

$$\begin{aligned} kp &= 10 \\ kd &= 1 \\ ki &= 8 \end{aligned} \tag{5}$$

These changes to the control values had a huge effect on the height stabilization. The overshoot was reduced 25% and the settling time was reduced to $10.3s$.

Following the modifications to the Height Controller, we will be experimenting with the other controllers to understand how to modify the response. From Figure 8, we would like the X-Y stabilization to be as quick as the Z stabilization.

We are executing these simulations to understand the controls of the model. Eventually, we will switch to developing the model in code for ROS. All of the control values will be different from what we use in Simulink. However, we will know how to tune the control values to work for the Crazyflie.

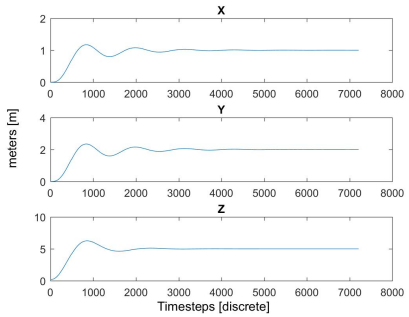

Figure 8: X,Y,Z Plots using control values in Eqn. 5

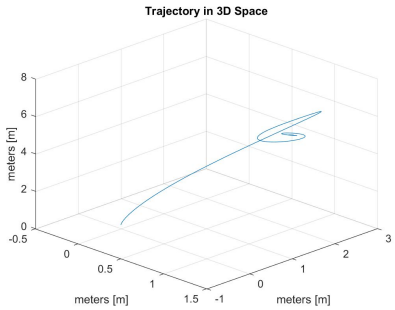

Figure 9: 3D Plot of Fig. 8

5 Research Problem 2: Implementing the Control of Crazyflie using ROS

5.1 Introduction of ROS

At the start of the project, no one in the department much experience with the ROS system. This means that extensive research was required in order to gain a deeper understanding of the ROS system. The main method that was used was reading a textbook [4] that Dr. Wang provided.

5.1.1 ROS Tutorial Textbook

The textbook went through step by step instructions on how to setup and work with a catkin workspace using ROS commands. Many of the commands used for navigating the environment are similar to the standard Linux bash commands. The textbook used ROS Indigo for its examples, so that is the version that we used as well.

5.1.2 ROS Basics

ROS is a very complex program that attempts to simplify the running of a program on a robotic system. ROS is run entirely through the command console on Linux. It consists of three of basic concepts:

- Packages: Contain the information needed to compile executables.
- Nodes: Small modules of processes that usually do a few tasks as part of

the larger project. Nodes can communicate with other nodes on the same computer or nodes on a different networked computer through the Master.

- Master: Provides the naming and registration services for the Nodes in the ROS system. The Master allows the different nodes to find each other so they can communicate.

In simple terms, nodes do simple processes like fetching and processing image data. Multiple nodes work together through the master to do much more complex tasks, like flying a drone.

5.1.3 Honig Paper

The "Flying Multiple UAVs Using ROS" paper[5] is a helpful paper that dealt with creating a swarm of Crazyflies. The only problem is that the paper is designed for flying more than 20 Crazyflies with an expensive VICON camera system for localization. We will be only flying 3 or 4 Crazyflies using the cheaper Kinect 360 cameras. However, the GitHub for the paper had all of the demo files that were used to test flying the Crazyflie with the PS3 remote. The ROS Textbook[4] used the files from the paper's GitHub that were designed for ROS Indigo.

5.1.4 Complications

For this project, one of the main benefits of ROS is also one of its pitfalls. The ability of ROS to be modular is touted as one of its best aspects. However, when the modular capability is abused, programs can become confusing and nearly impossible to follow. This is the case with the demo programs described earlier. There are so many different aspects of the Crazyflie that need are being controlled through various dependency files that it is extremely difficult to follow.

Possibly the toughest task for this project on the ROS side of things will be figuring out exactly how to edit the programs to fit our needs. The first goal is to be able to write a short program that can send information to the Crazyflie through the Crazyradio PA. This will also require us to figure out how to remotely flash the firmware on the Crazyflie.

5.2 The Basic Program Structure of Flying a Crazyflie Using a Remote

Using the files from the Honig GitHub, we were able to successfully fly the Crazyflie using the Crazyradio PA and a PS3 remote. The demo files are

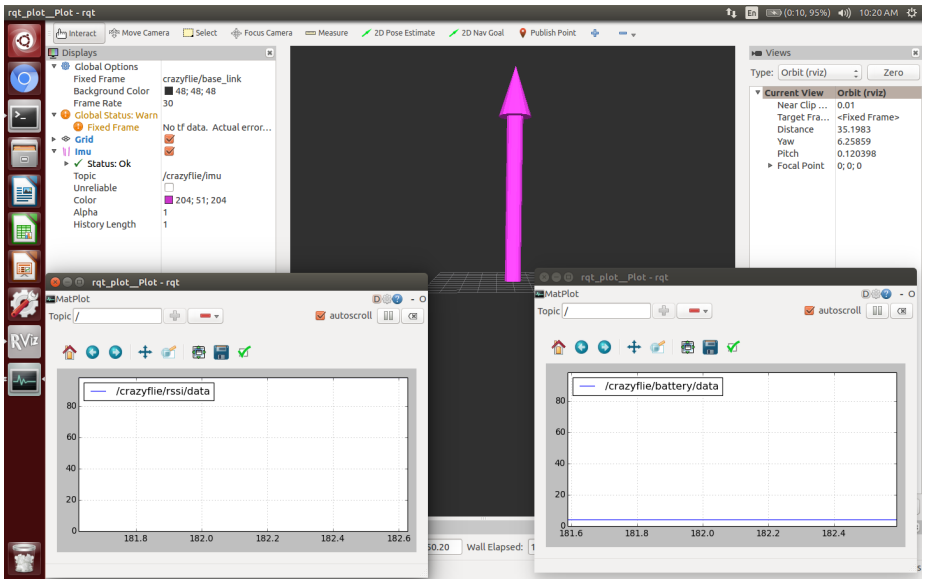

Figure 10: Screenshot of ROS running with the Crazyflie connected. The IMU status is shown on the graph in the center with the pink arrow. Other information, like battery level, is displayed in the other graphs.

extremely complex with multiple dependencies spreading through multiple directories in the catkin work-space that is made using ROS.

When the demo is run through ROS, many different nodes are created. An RViz window is opened that shows the status of the on-board IMU. Figure 10 on page 14 shows what is displayed while running the demo.

6 Research Problem 3: Localization of Crazyflie using Xbox 360 Kinect and LOCO Positioning System

6.1 Functionality of Kinect Sensor

The kinect sensor uses multiple camera sensors to measure position data. The first two camera sensors measure 3D distance data by tracking the light pattern disruptions as shown in [6], by using a known pattern of light. The third camera on the front uses color and depth from focus shown in [6] , and although the exact method is undisclosed, it is based on the structured light principle. The kinect uses these sensors together to form a 3D position of an object. For our

Figure 11: Xbox 360 Kinect Module

uses we will get the 3D position of our Crazyflie and feed this information to our PD controller.

6.2 The Basics of Image Processing in ROS Environment

There are repositories that exist to connect our Kinect to the ROS enviornment, feeding the 3D position information to our ROS controller. The sensor, working with our ROS controller, will help to control our Crazyflie. The algorithm referenced in [4] uses the kinect sensor and creates a color threshold in our ROS enviornment. This threshold tells the ROS simulator where in 3D our Crazyflie will be, and will be used to manipulate where the Crazyflie will be moved to in 3-space. The threshold code will also allow us to use multiple Crazyflies at the same time with different color markers.

6.3 The Basic Idea Using Kinect for Localization

The algorithm suggested by the ROS book uses color thresholding to sense where the Crazyflie is in the frame of the Kinect. The method reads a specified color from the Kinect image data and has a certain distance that the color has to be constant (within a certain tolerance of color). The Kinect can output this information as a 3 dimensional variable to the controller. The thresholds need to be set according to the specific Kinect model and the size and color of the marker that will be set on the Crazyflie itself.

We have been provided with 3 Xbox 360 Kinect modules as shown in Figure 11. The ROS book references the Kinect V2.0 for its superior sensors and frame rate, which will be a concern with our Kinect 360 modules. This will have to be overcome during our implementation of the algorithms mentioned above. There are multiple Git Repositories that reference the Kinect V2.0, so we should be able to convert that code to work with our Kinect 360 modules.

The next step in our research will be to implement multiple Crazyflies in the

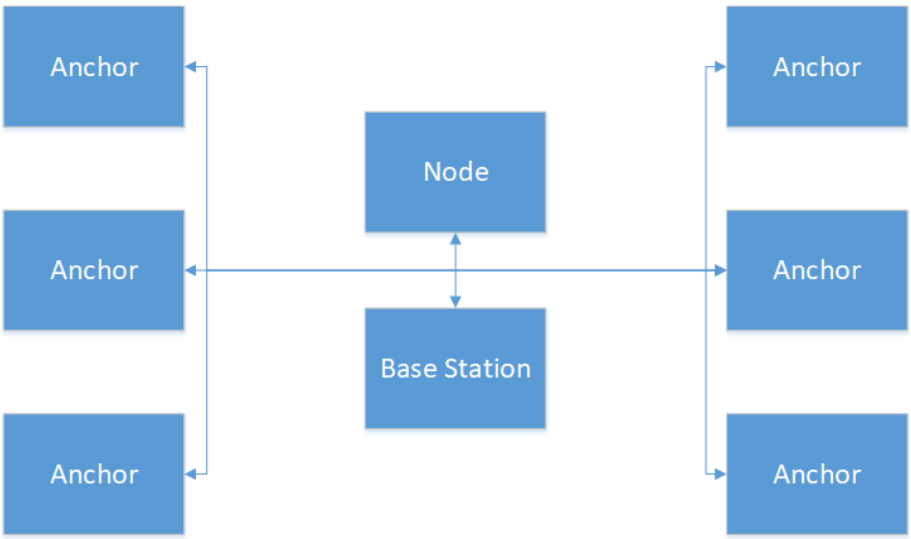

Figure 12: LOCO System Diagram

same test. The algorithm can work to have multiple colors, meaning that each Crazyflie will have its own color marker above it to feed back position information for all of the Crazyflies in the frame of the Kinect. We may also be able to use multiple of the same colors and have the Kinect sense that the distance between each color marker means that there are multiple separate Crazyflies even though they have the same color marker.

The Kinect modules will be set up on stands keeping them about a meter off the ground. This will create a more perfect square frame that we can use to control the Crazyflies. We will use multiple Kinect modules over time to get better position information for our control algorithms. We will set up a square frame where we want the Kinect modules on different sides of the square to get more complete position information, and to allow for yaw control.

6.4 LOCO Positioning System

The LOCO Positioning System operates like a small indoor GPS system. Anchors around the room act as position reference markers in the 3D space. Nodes are tracked through the 3D space, one of which is on the Crazyflie.

This system may be able to be used in place of the Kinect Module. Eventually we may be able to use the Kinect and LOCO system in tandem for some redundancy. The LOCO system is accurate to 10cm which is accurate enough for our project as we do not plan on doing extremely tight maneuvers.

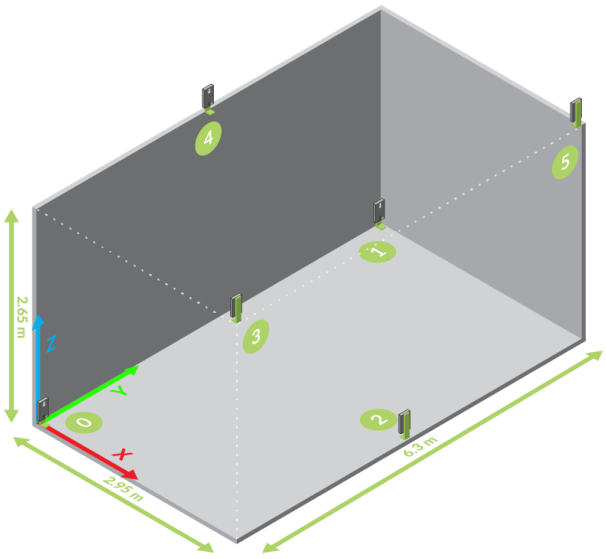

Figure 13: LOCO Setup Diagram

We will have to research the capability to track multiple Crazyflies. Right now the system can only track one Crazyflie. So we will have to purchase more LOCO Crazyflie decks to incorporate more Crazyflies.

7 Research Problem 4: System Integration and Formation Control Implementation

7.1 System Integration

The LOCO Positioning System and the Kinect 360 Module can detect the positioning of the Crazyfiles. The ROS environment on the base station will accept inputs from the Crazyflies, the LOCO system and the Kinect. The Crazyflies will run the control algorithm on their on-board chips. The Kinect and LOCO systems will provide localization data to the base station. The base station runs the cooperative control algorithm and provides localization updates to the Crazyflies.

7.2 Formation Control

We will be looking to implement cooperative control theory to command the positioning of the Crazyflies. The first step will be to simulate multiple Crazyflies in Simulink. This should be fairly simple to implement. The current model shown in 3 would be a subsystem within the new model. Each subsystem would then represent one Crazyflie. The cooperative control algorithm would be running outside all the Crazyflie models, feeding each Crazyflie model with its updated location coordinates.

8 Parts List and Work Plan

8.1 Parts List

1. Bitcraze Components
    - 6 x Crazyflie 2.0
    - 3 x Crazyradio PA
    - 1 x Z-ranger Deck
2. Xbox Components
    - 3 x Xbox 360 Kinect
    - 3 x Xbox 360 Kinect Stand
    - 3 x Xbox 360 Kinect Power Supply
    - 1 x Xbox ONE Kinect
3. LOCO Positioning System
    - 6 x LOCO Anchors
    - 6 x Anchor Power Supply
    - 6 x 3D Printed Anchor Brackets
    - 1 x LOCO Crazyflie Deck
4. Laptop running Ubuntu 14.04 Trusty

8.2 Schedule

November 2017:

- **28:** Final Proposal and Presentation Due

December 2017:

- **5:** LOCO system setup
- **7:** Website with Deliverables Due

January 2018:

- **31:**
  - Single Crazyflie Control Operational
  - Multiple Quadrotor Model & Simulations

February 2018:

- **16:** Project midpoint check
  - Kinect control operational
  - LOCO system operational

March 2018:

- **29:** Final Report draft due

April 2018:

- **10:** Student Expo
- **26:** Project presentation ready

May 2018:

- **1:** All deliverables due

8.3 Work Division

Bryce Mack has being developing the model and running simulations. He will continue to do work on tuning the controllers so that we have the smoothest flight for implementing the hovering algorithm. He will also be focusing on developing a new model to run simulations for multiple Crazyflies. Along with

running simulations, he will create videos for future demonstration. In addition to simulations, he will record simulation data like settling time and overshoot to determine if the control meets our requirements. The versatility of the model to change the control inputs makes this very easy. Bryce will also be working on the website. The website should be presentable and easy to navigate. All the deliverables will be there as well.

Chris Noe has been studying ROS and its functionality. He has been researching how to get the ROS environment to communicate with the Crazyflie 2.0. He has been looking into the demo code in order to find out how the demo works so it can be modified to suit our needs. He will also be researching the LOCO positioning system that Dr. Wang decided we would introduce to our project. The LOCO system has detailed setup instructions on the Bitcraze website. Chris will be following their instructions to get it setup for experimentation. Ideally, we will be able to integrate ROS with the LOCO system so that we can implement our control algorithm and recreate the simulations.

Trevor Rice has also been studying ROS for our initial development. He has been researching how to incorporate a Kinect 360 into an ROS workspace. He will be researching and documenting which libraries and packages are needed to setup the ROS workspace. He will be researching more on how to use the Kinect 360 cameras to detect position. Trevor will be setting up the space in the lab that we will be using for physical experimentation. There will need to be a clean area setup because there shouldn't be anything to interfere with the Kinect's cameras and sensors. A clean area will prevent the possiblilty of a false detection of the Crazyflie.

References

[1] Pedro Castillo, Rogelio Lozano, and Alejandro Dzul. "Stabilization of a Mini Rotorcraft with Four Rotors". In: *IEEE Control Systems* (2005).

[2] Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB®*. Springer, 2011. ISBN: 9783642201431.

[3] Zachary T. Dydek, Anuradha M. Annaswamy, and Eugene Lavretsky. "Adaptive Control of Quadrotor UAVs: A Design Trade Study With Flight Evaluations". In: *IEEE Transactions on Control Systems Technology* 21.4 (2013).

[4] Carol Fairchild and Thomas Harman. *ROS Robotics By Example*. PACKT Publishing, 2016.

[5] Wolfgang Honig and Nora Ayanian. *Flying Multiple UAVs Using ROS*.

[6] Winjun Zeng. "Microsoft Kinect Sensor and Its Effect". In: *IEEE Computer Society* ().