Distributed Control of Crazyflies

Bryce Mack, Chris Noe, and Trevor Rice
Advisors: Dr. Jing Wang and Dr. In Soo Ahn

BRADLEY University

April 28, 2018

Overview

- Introduction
  - Motivation
- Modeling and Control Design
- Experimental Implementation
  - Using Kinect 360 for localization and control
  - Using Loco Positioning System for localization and control
- Conclusions
- Q & A

Introduction

Motivation

- UAVs have attracted significant attention in both industry and military in recent years
  - Reconnaissance
  - Cooperative exploration for search and rescue missions
  - Environmental observation

Motivation

- Cooperative transportation

Motivation

- Entertainment

Motivation

- Surveillance and Monitoring

Objectives

- The challenges are:
    - How to deal with information sharing based on sensing/communication among individual UAVs
    - How to design simple yet efficient local control strategies for each UAV
- The overall goal of this project is to:
    - Design practically implementable distributed control algorithms for UAVs
    - Implement algorithms using an agile nano quadcopter, the Crazyflie

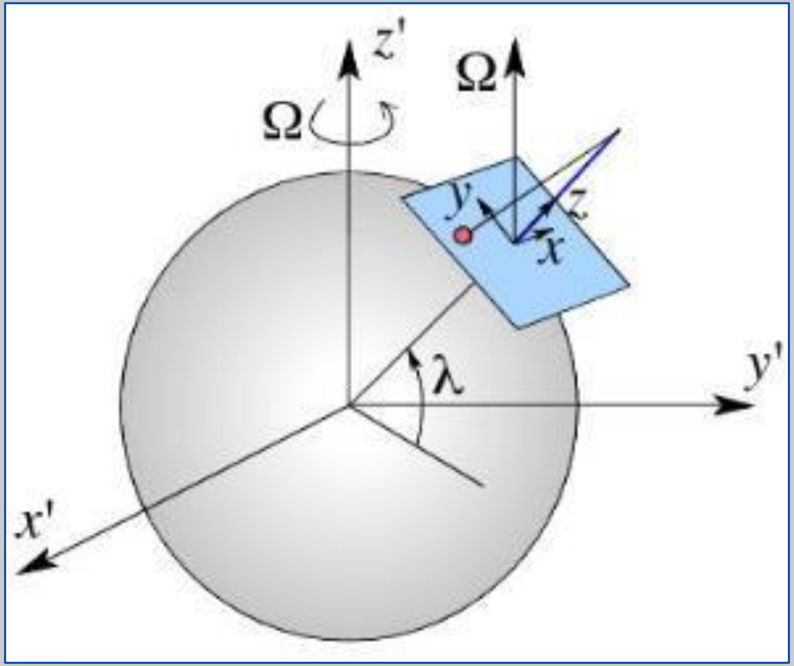
Objectives

- Use the Kinect 360 camera for localization and stabilizing control of Crazyflie
- Use the Loco Positioning System for localization and stabilizing control of multiple Crazyflies
- Design and implement control algorithms for Crazyflie following various trajectories
- Design and implement formation control algorithms for multiple Crazyflies

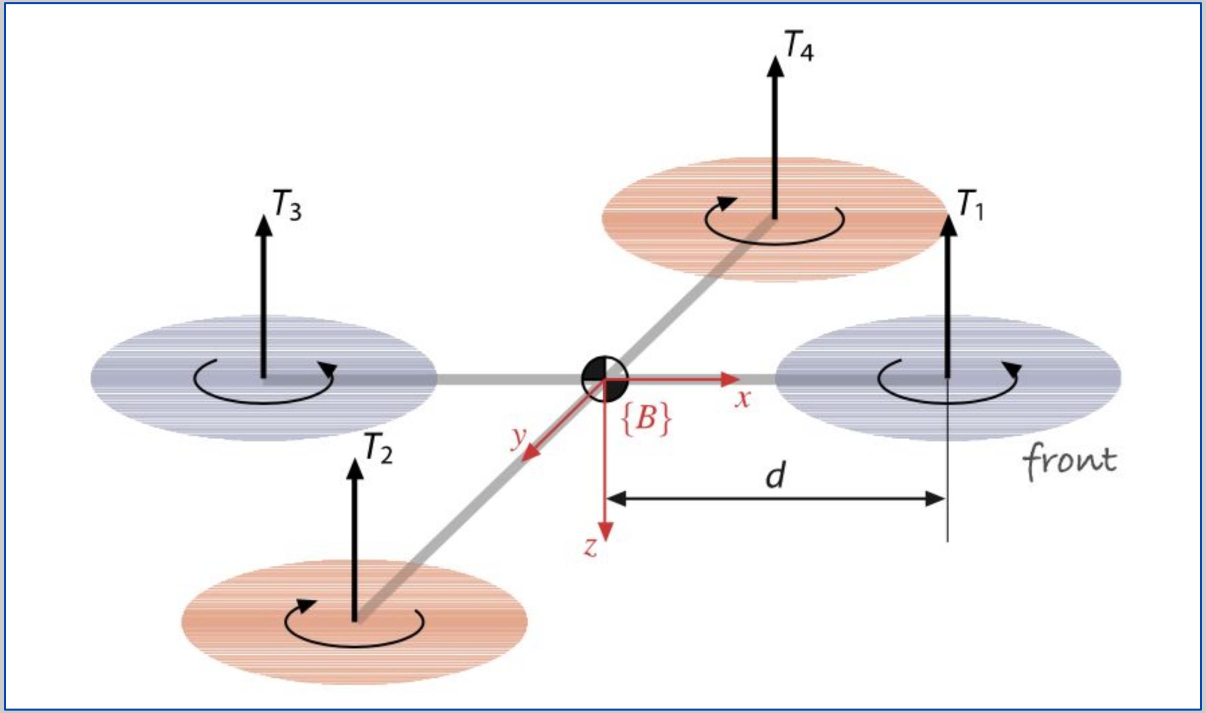
MODELING AND CONTROL DESIGN

Quadrotor Coordinate System

- The inertial frame designates Z to be any direction coming out of the earth
- The body frame of the quadrotor designates Z to be into the earth



Quadrotor Body Frame



Quadrotor Model

- *Adaptive Control of Quadrotor UAVs: A Design Trade Study With Flight Evaluations*
  - By Zachary Dydek
- Using small angle approximations Eqn. 1 becomes Eqn. 2

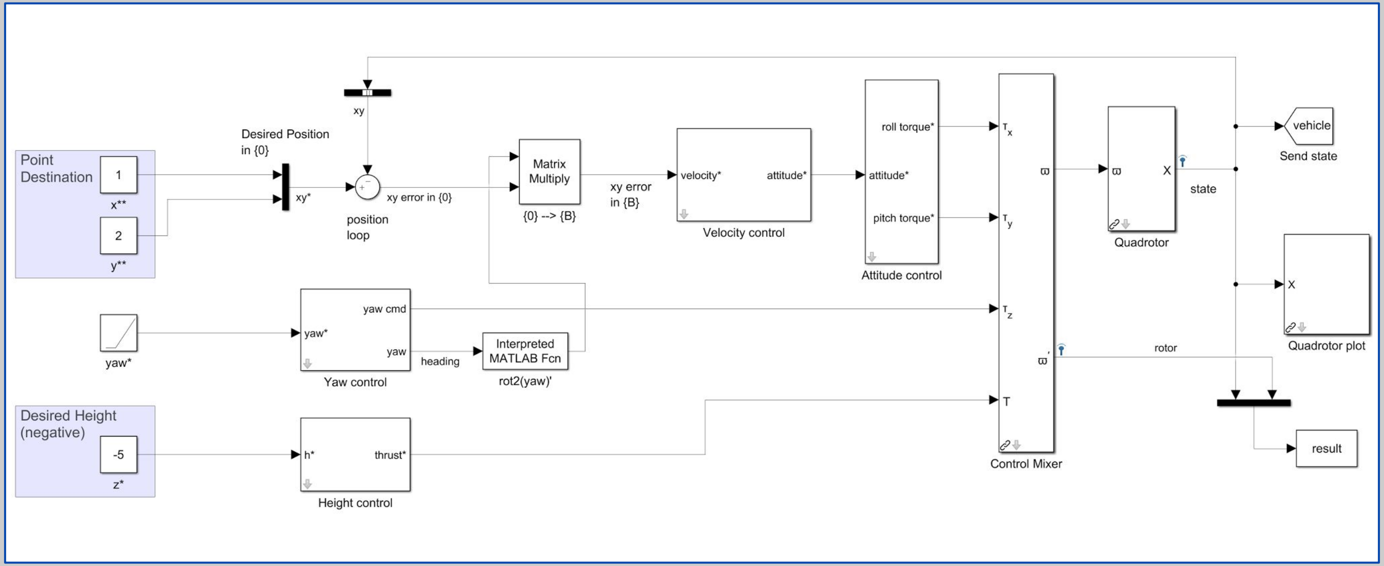
$$
\begin{aligned}
\ddot{x} &= (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)\frac{U_1}{m} \\
\ddot{\phi} &= \dot{\theta}\dot{\psi}\left(\frac{I_y - I_z}{I_x}\right) - \frac{J_R}{I_x}\dot{\theta}\Omega_R + \frac{L}{I_x}U_2 \\
\ddot{y} &= (\cos\phi \sin\theta \sin\psi + \sin\phi \cos\psi)\frac{U_1}{m}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

$\rightarrow$

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

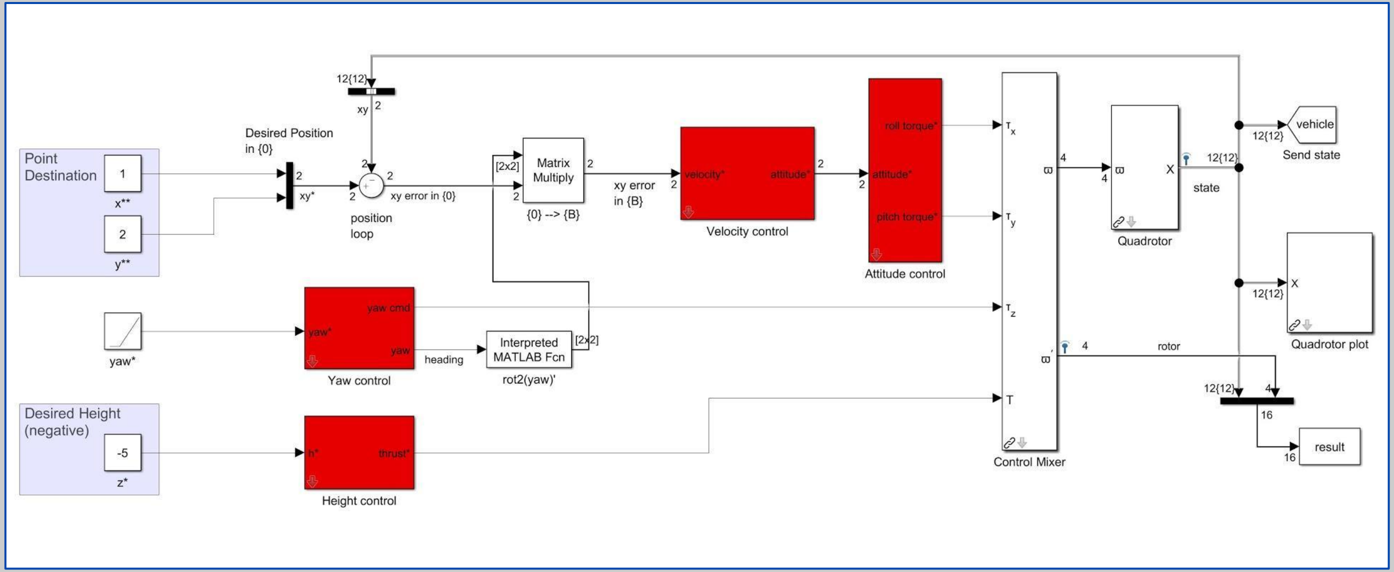
Simulink Modeling

- Installed MATLAB Robotics, Vision and Control Toolbox developed by Peter Corke
- Explored the Quadrotor model that they created



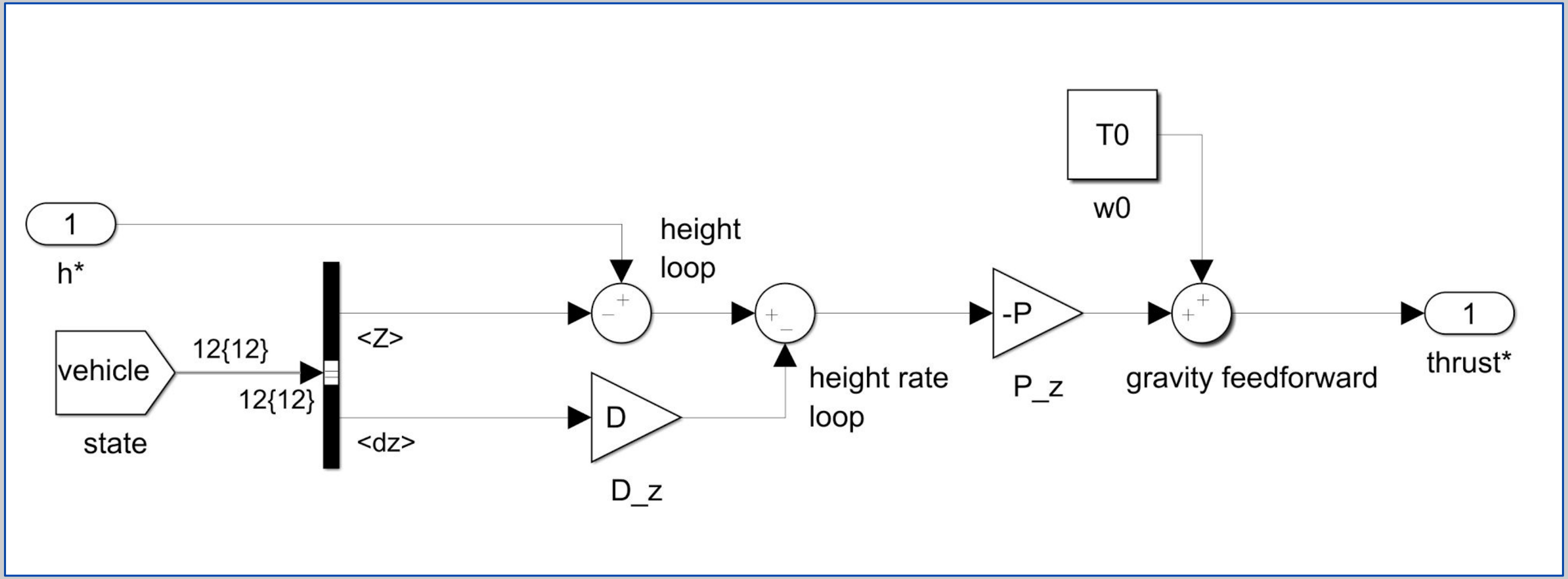
Hierarchical Control Strategy

- High-level control for waypoint generation
- Low-level control for Height, Velocity, Yaw and Attitude



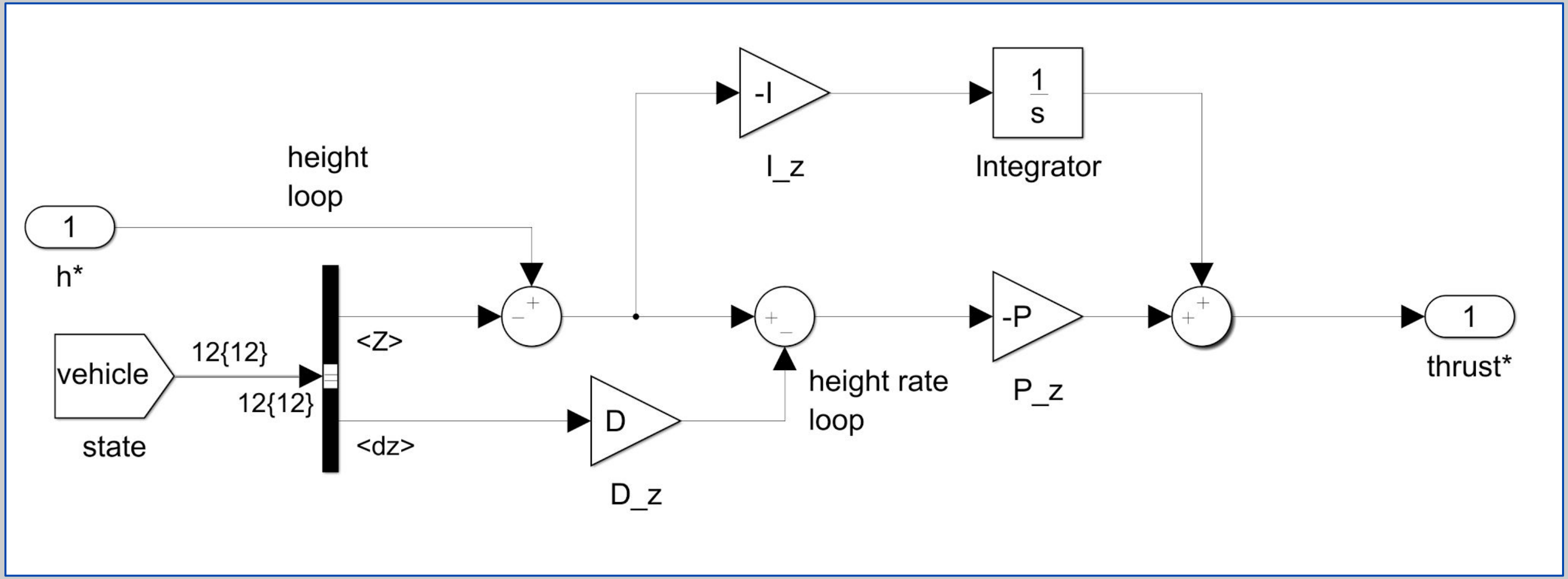
Altitude Control

- Old model: PD + a feed-forward thrust constant



Height PID Controller

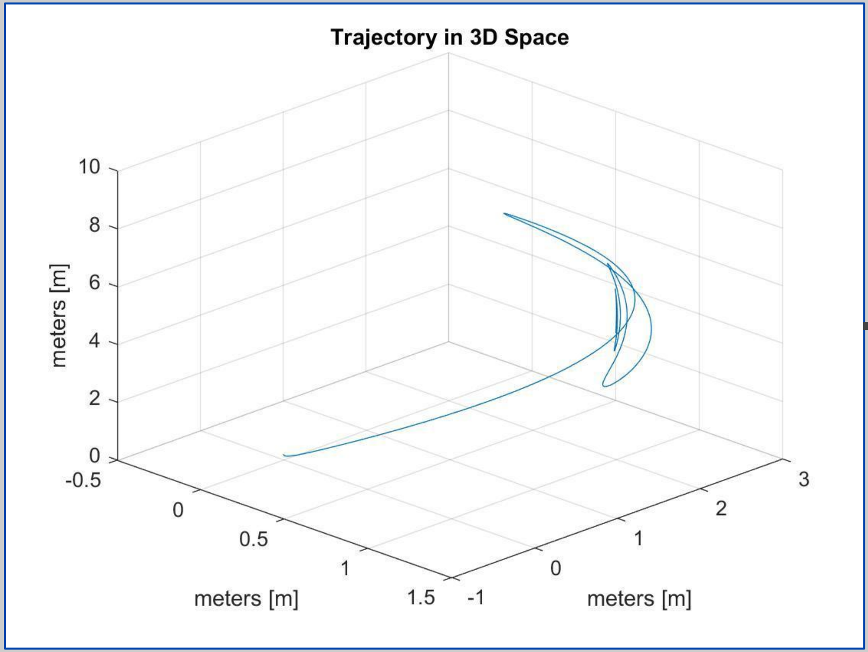
- Design PID which replaces feed-forward term

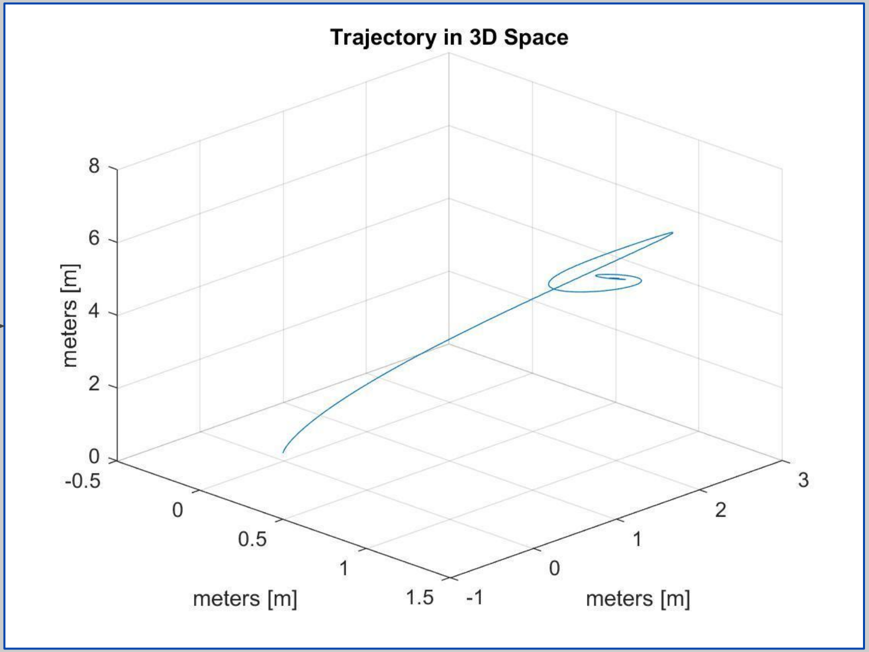


PID Tuning

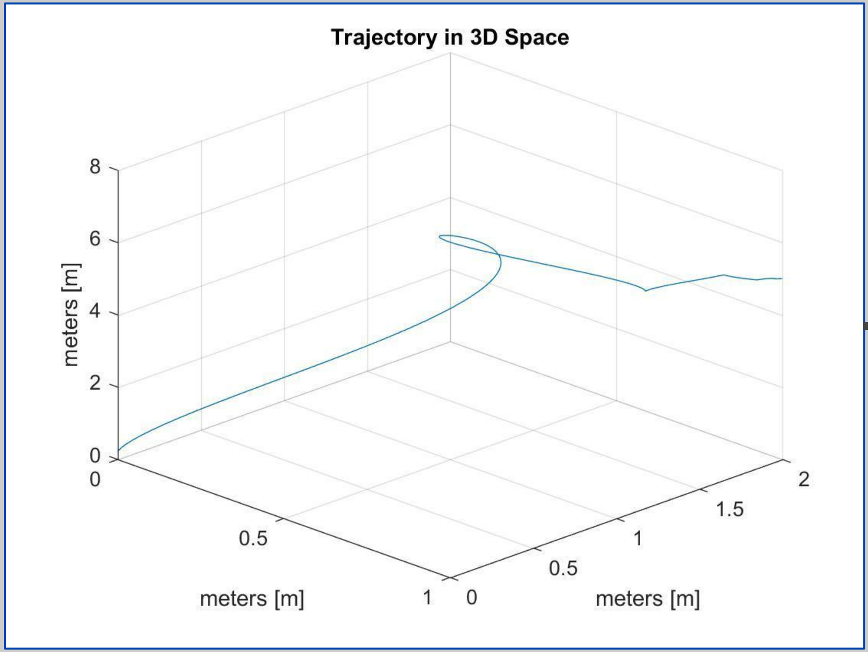
- Initial Height PID Gains
  - KP = 4
  - KD = 1
  - No KI
- Tuned Velocity Gains
  - KP = 12
  - KD = 5
  - KI = 0.6
- Initial Velocity PID Gains
  - KP = 0.1
  - KD = 0.2
- Tuned Velocity Gains
  - KP = 0.02
  - KD = 3.4

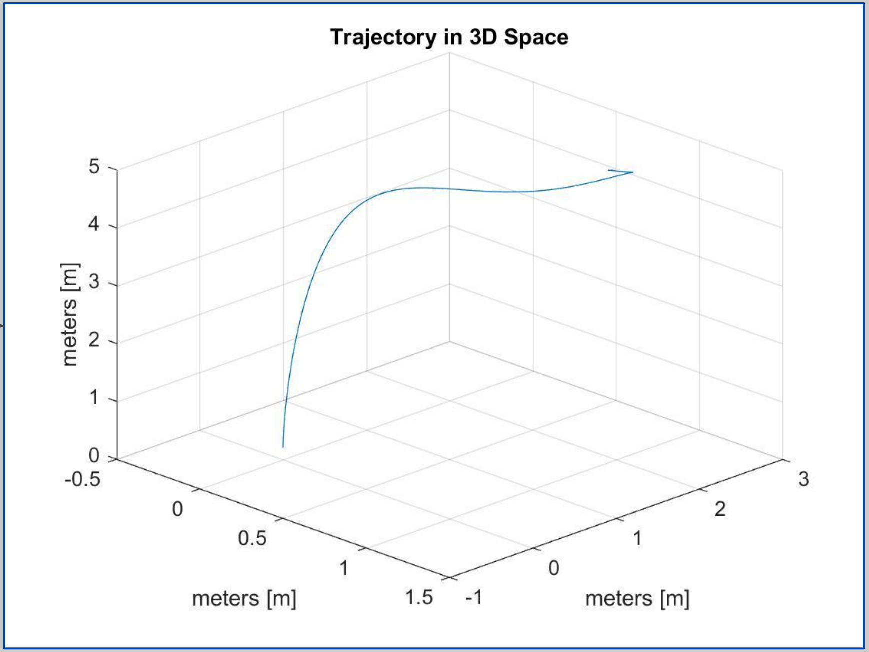
Adjusting Height Control





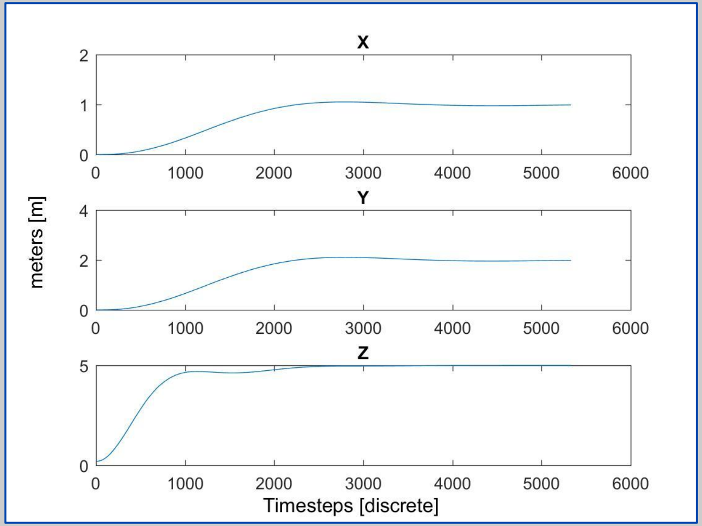
Adjusting Velocity Control





Simulation Results

- Tuning the controllers allowed us to reduce the overshoot to 6% for X and Y
- We decided for Z to have 0% overshoot
  - Critically damping the system
- We don't want the crazyflie ever crashing into a ceiling was our reasoning
- Z is able to settle within 6.5 seconds



Trajectory Control

- Circle
  - sine and cosine inputs to simulate a circle
    - 3sin(t/8)
    - 3cos(t/8)
- Figure 8
  - 2 sin inputs that create a figure 8
    - 3sin(t/10)
    - 3sin(t/20)
- Square
  - 4 step inputs
  - Each activating after ‘x’ seconds

Circle Trajectory

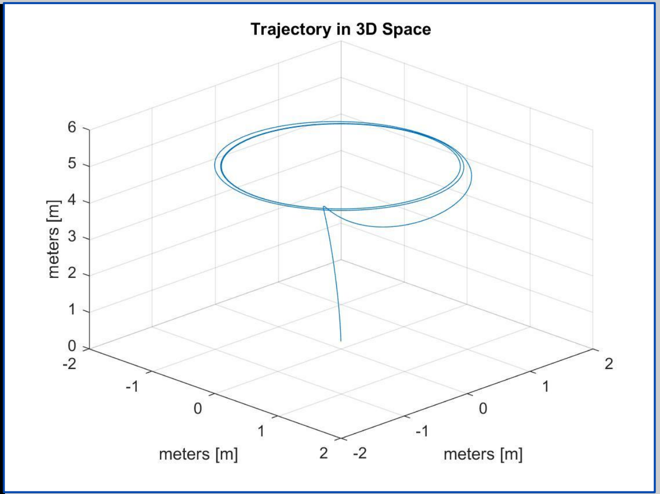
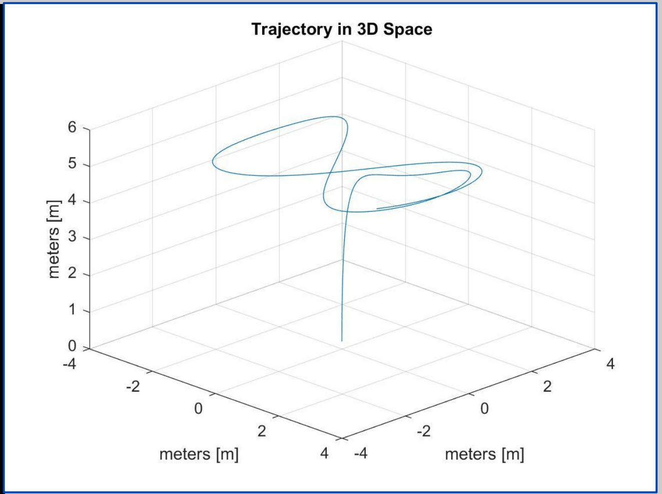
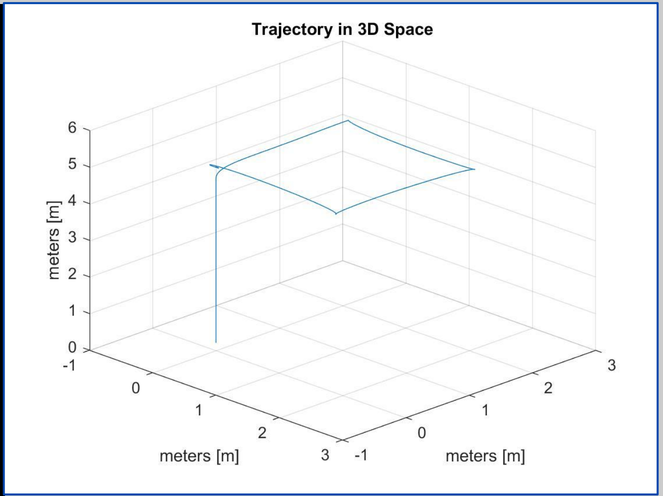


Figure 8 Trajectory



Square Trajectory



Experimental Implementation

Parts List

Bitcraze Components

- 6 x Crazyflie 2.0
- 3 x CrazyRadio PAs
- 1 x Z-Ranger Deck

Xbox Components

- 3 x Xbox 360 Kinects
- 3 x Xbox 360 Stands
- 3 x Xbox 360 Kinect Power Supplies

Loco Positioning System (LPS)

- 6 x LPS Anchors
- 6 x Anchor Power Supplies
- 6 x 3D Printed Anchor Brackets
- 5 x LPS Crazyflie Decks

Laptop Running Ubuntu 14.04 Trusty

Total Cost: $2585

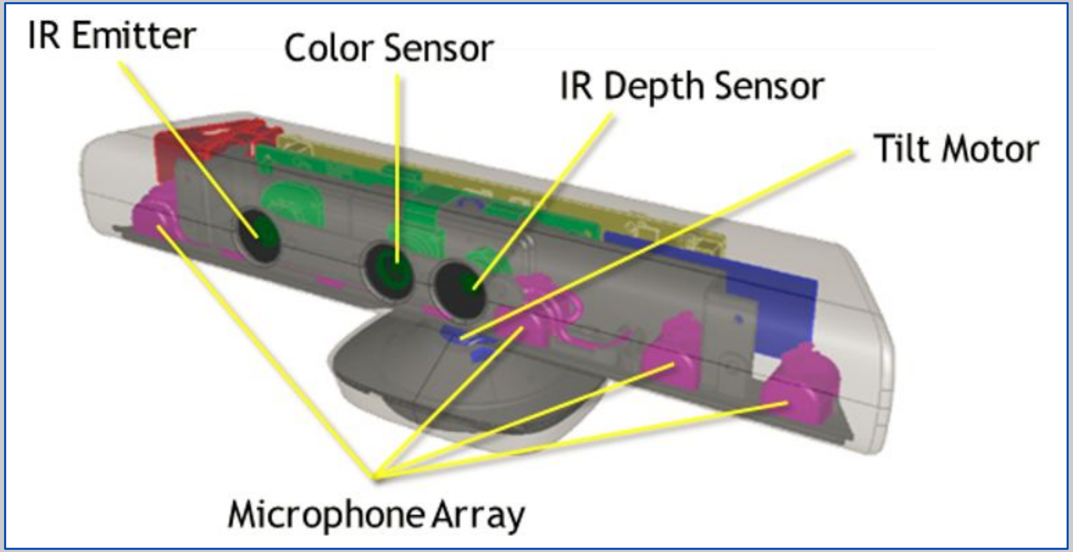
Using KINECT 360 for Localization and Control

Kinect 360

- Kinect has 3 I/O devices: cameras, audio, and motors
- Only used camera functions, not audio output or motor input
  - The camera has 2 outputs to the python script
    - RGB camera - Video
    - IR Blaster and monochrome CMOS sensor - Depth
- Kinect captures 640 x 480-pixel resolution at 30 FPS

Kinect Localization

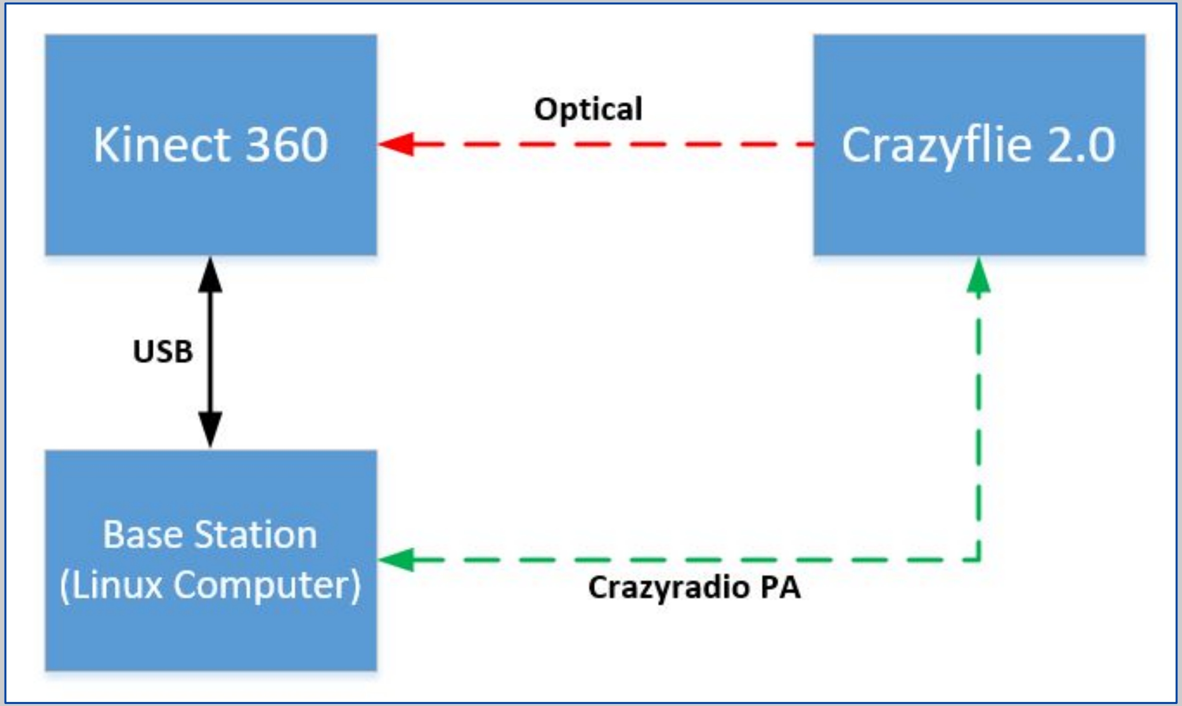
- Kinect localizes the Crazyflie using two devices, the RGB camera and the IR camera
  - IR camera provides depth
- RGB camera identifies the marker set on the Crazyflie
  - Red tape



Kinect Libraries

- Discovered that python library libfreenect2 does not recognize the Kinect 360
- Libfreenect, older library, successfully communicates with Kinect 360
- Libfreenect and Crazyflie-Clients-Python library bridged communication between the Kinect and the Crazyflie
- Scripts were updated to output localization data to .csv file
- Tuned PID values to test for better control
    - Decided original values were optimal

Kinect System Diagram



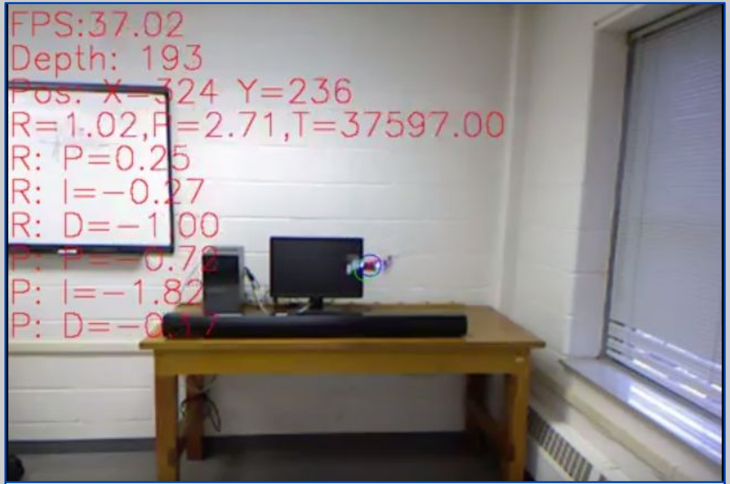
Kinect Workspace

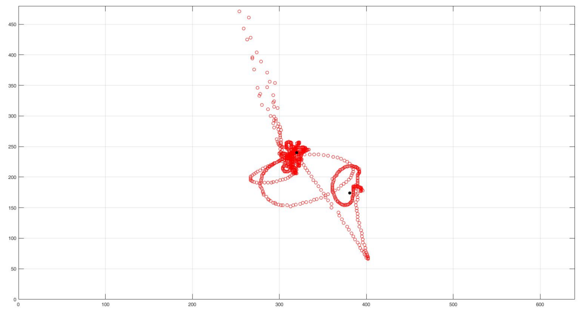
- Dedicated space in Robotics Lab for Crazyflie operation
- Kinect operating ranges
  - Minimum distance: 0.5 m
  - Maximum distance: 4.5 m
  - Height: 0-3 m
- Flight area defined by the line of sight of the Kinect
  - The flight space needs to be “visually clean”
    - Red in the frame would cause a false positive

Testing Results

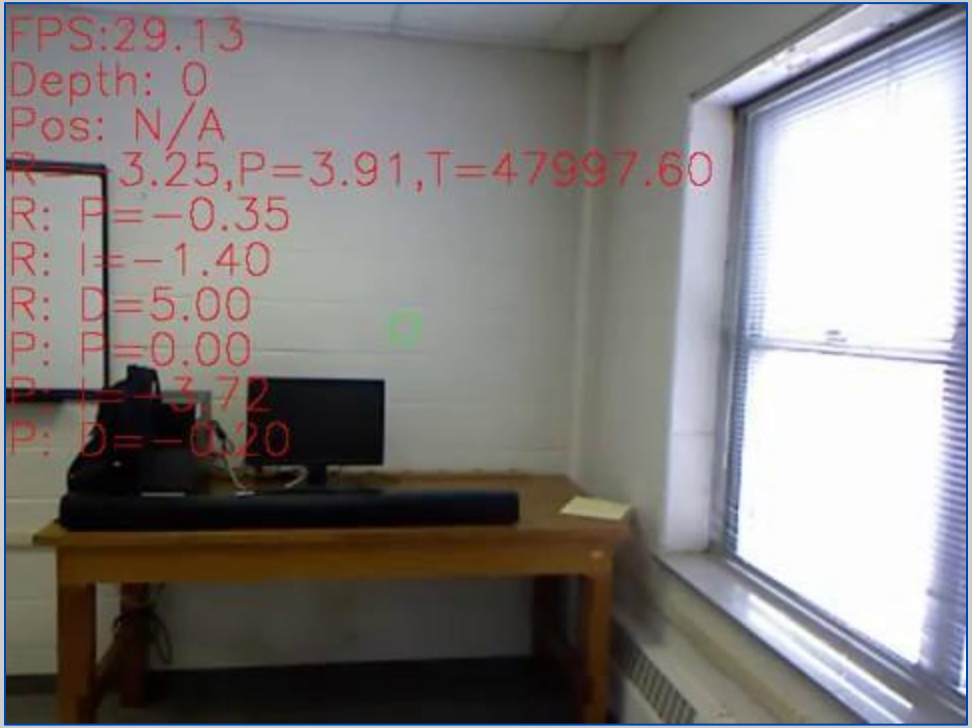
- Photo is a screenshot of the video stream shown when controlling the Crazyflie
- Text overlay displays a live feed of PID control values, position, depth, and thrust

- Graph shows X and Y output data from one of the flights plotted in MATLAB





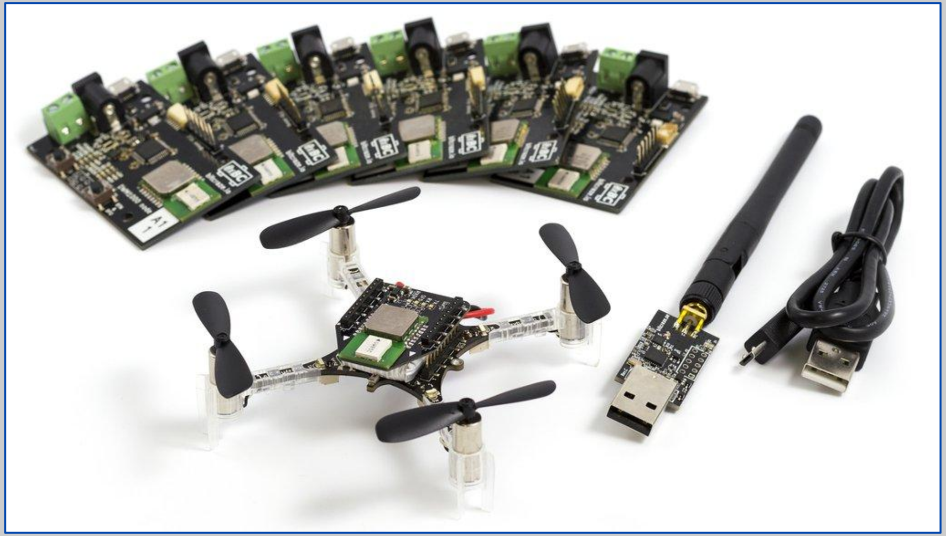
Testing Results



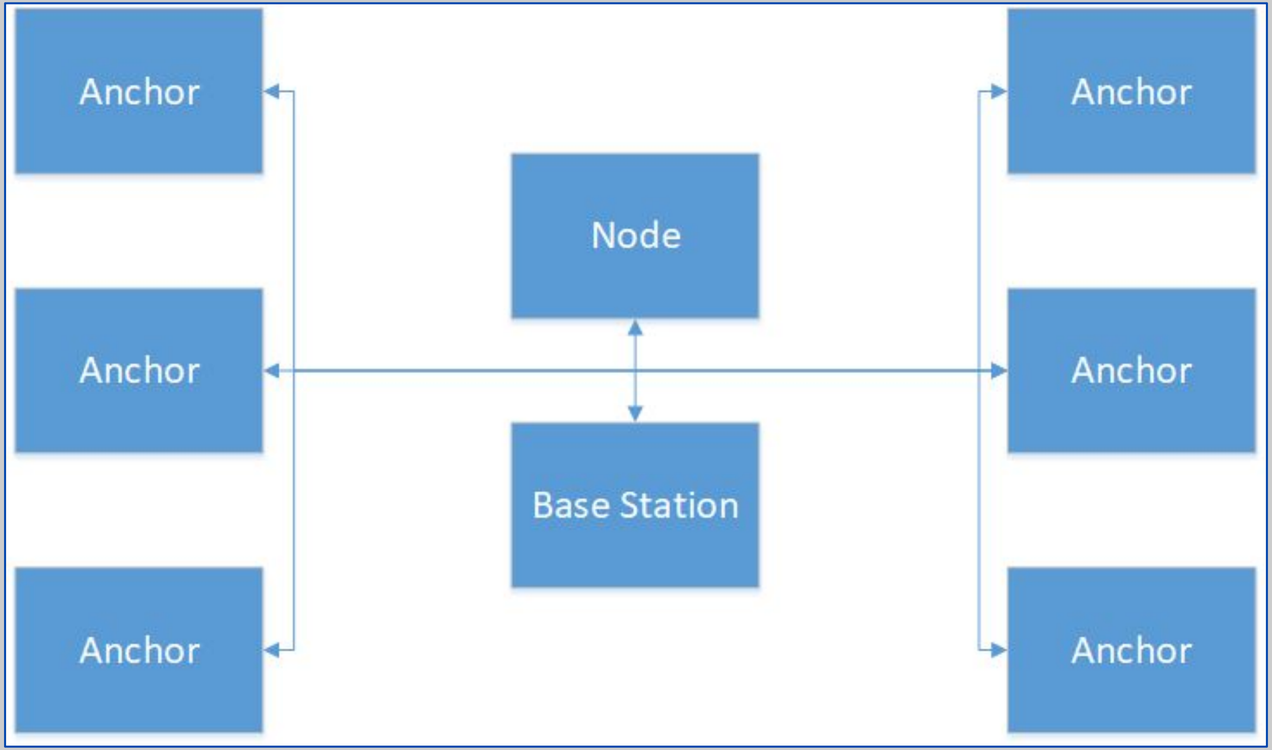
Using Loco Positioning System for Localization and Control

Loco Positioning System (LPS)

- Set up as an “indoor gps system”
- 6 anchors set up in our space
- A tag (deck) is placed on each crazyflie
- Uses 2.4 GHz pings to estimate location in 3D space



LPS System Diagram

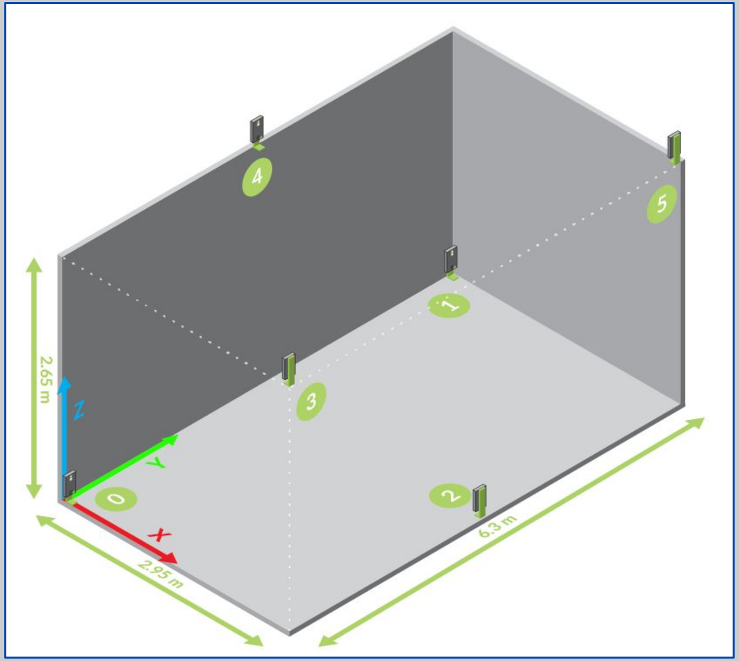


LPS Modes of Operation

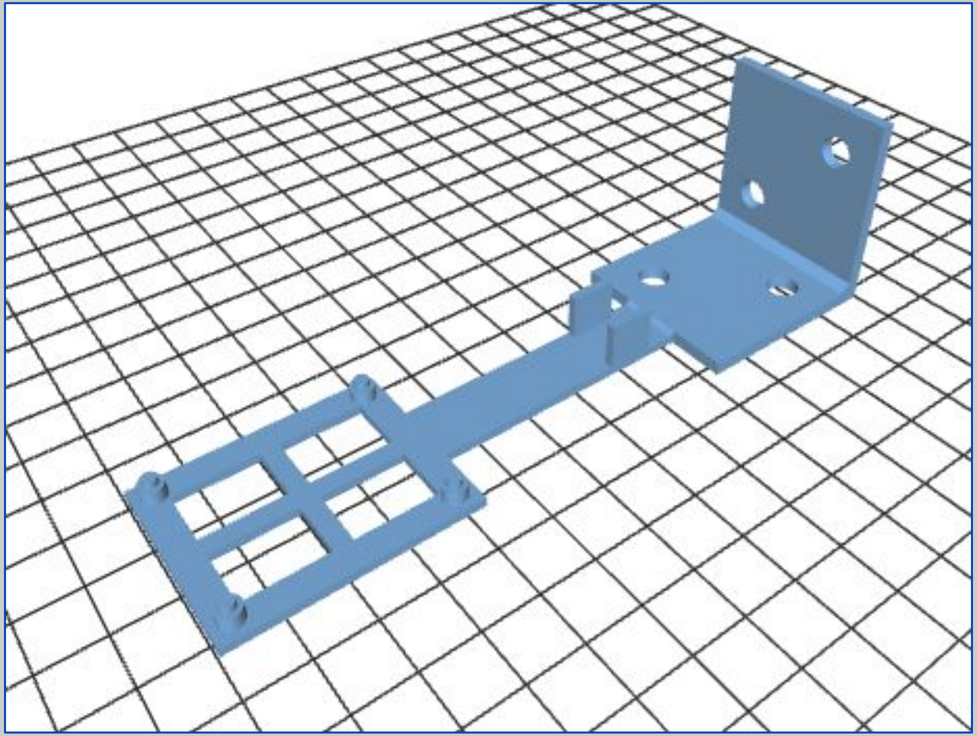
- Two-Way Ranging:
  - Anchors and Crazyflie both send out pings
  - More accurate mode
  - Limited to 1 crazyflie
- Time Distance of Arrival (TDoA):
  - Only the anchors send out pings
  - Slightly less accurate
  - Can be expanded to multiple Crazyflies

LPS Setup

- 3D printed brackets used to keep space between the anchors and solid surfaces
- Accurate measurements must be taken of the locations of the anchors
- Used 6 anchors
  - Can be expanded to 8 anchors for greater accuracy



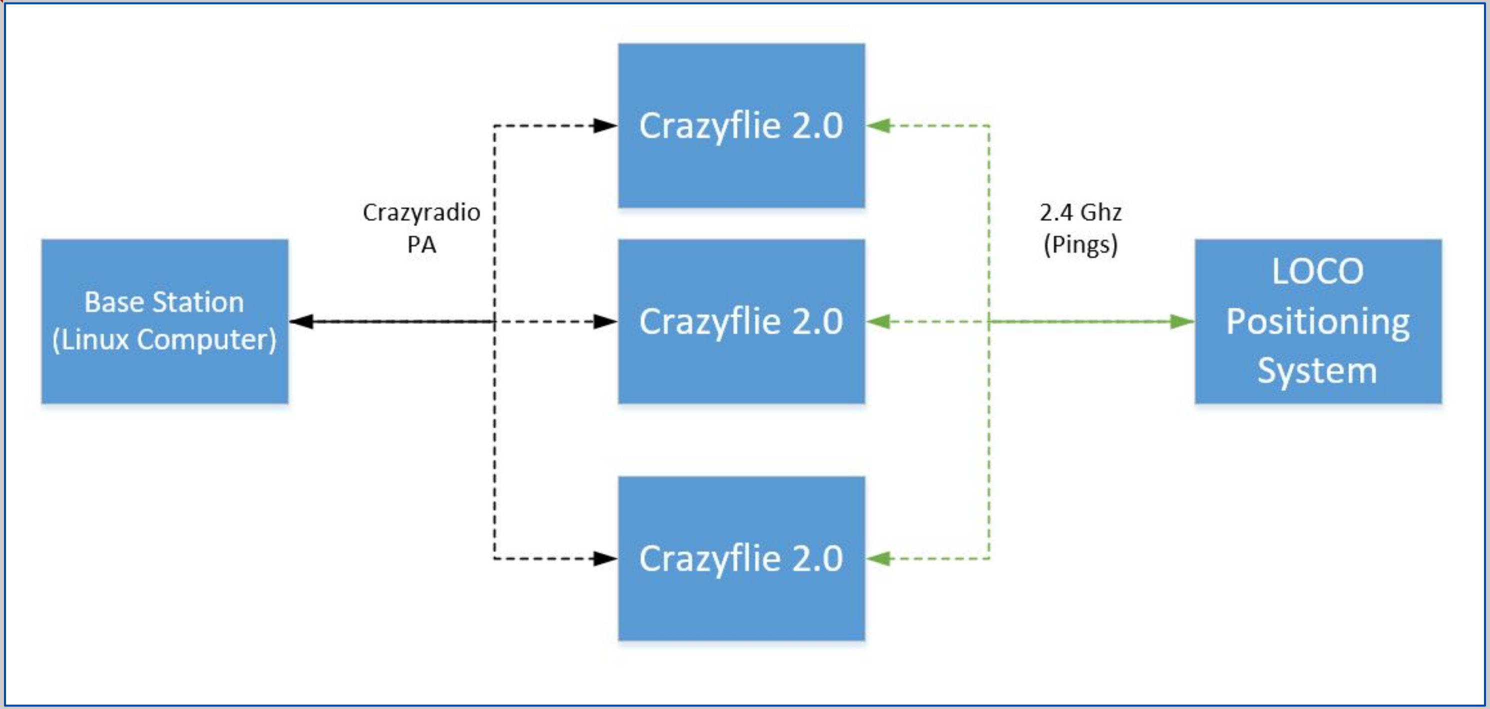
3D-Printed Anchor Brackets



Python Library Setup

- Initially tried using Ubuntu 14.04 virtual machine
  - Latency issues with library execution
  - More success using the Bitcraze virtual machine
  - Also tried in an Anaconda environment on a Windows machine
  - Setup in Virtualenv (virtual python environment) on laptop with Ubuntu 14.04 natively installed (Dr. Wang's PC for future use)

High-Level System Diagram

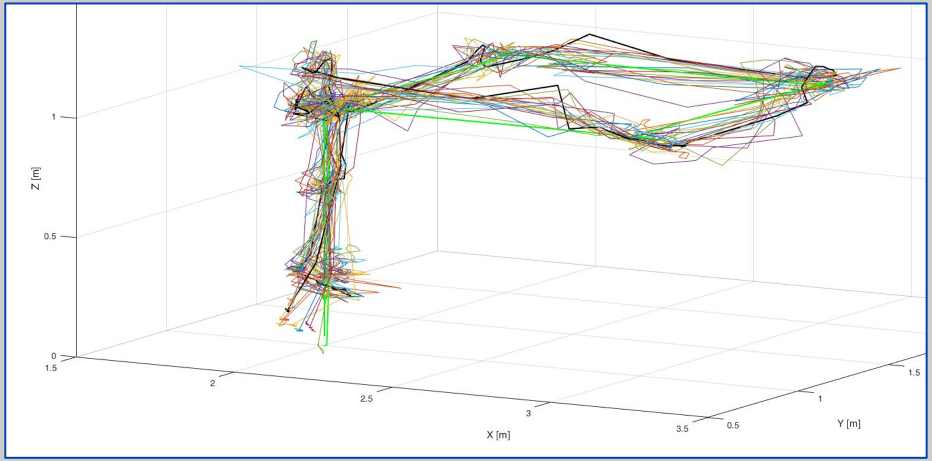


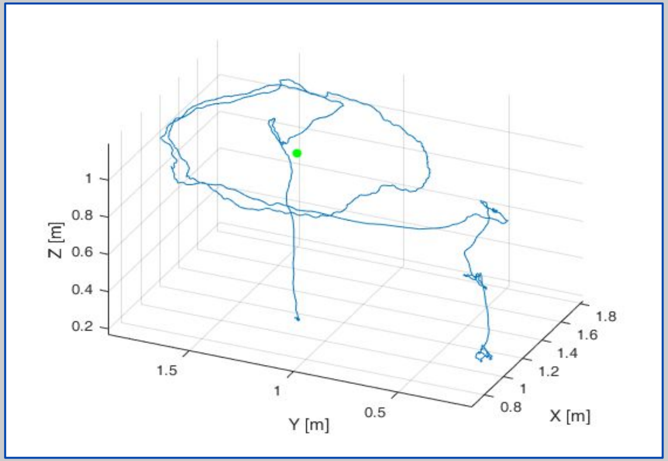
Python Library Development

- Library was developed to handle lower level control of the crazyflies (rotor speed, IMU readings, etc)
- We focused on higher level control functions
    - Sending position setpoints to the crazyflie
- Developed code, from examples, using the library to create formation flights
- Saved flight positioning data to .csv files
    - Later analyzed using MATLAB

LPS Results - 1 Crazyflie

- Started off programming hover sequence
- Proceeded to program square and circular trajectories
- Square Flight: within 20% of target height at 1m (0.8049m-1.19m)
    - Max takeoff overshoot of 26.9% (1.269m)

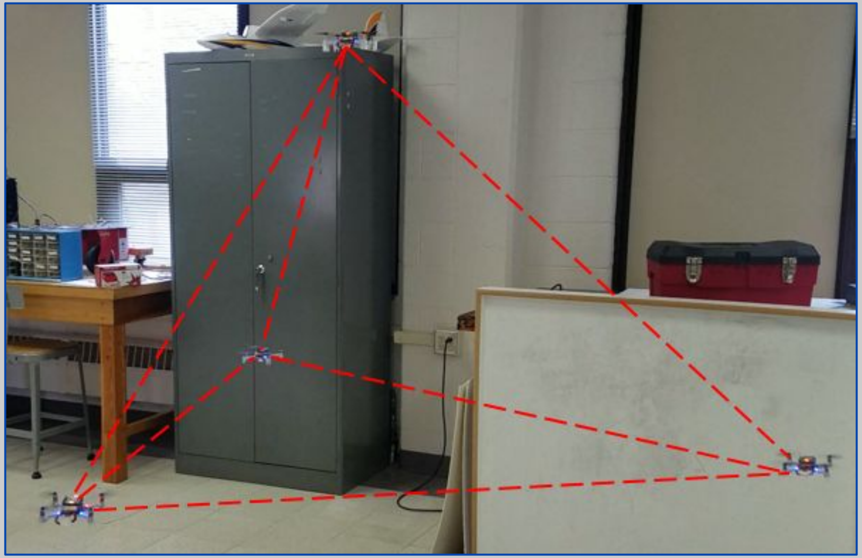




Circular Flight

LPS Results - Multiple Crazyfies

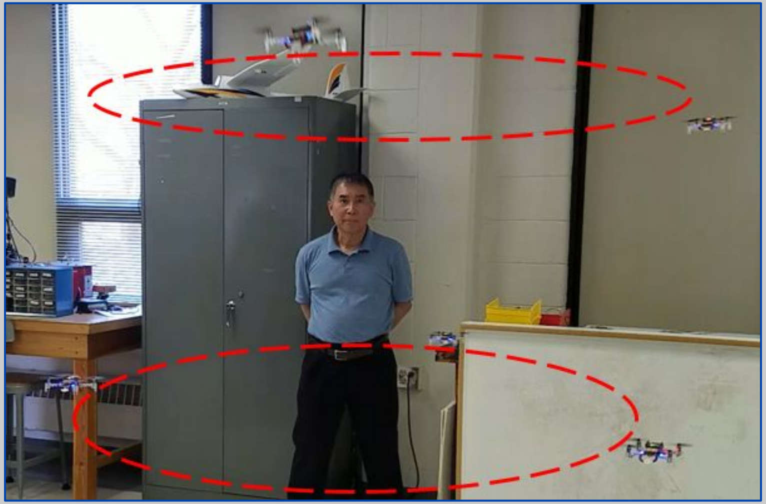
- Used library's parallel threading function to expand to multiple Crazyflies
- Flew up to 4 Crazyflies in "Pyramid" Formation
  - One drone hovering at 1.75m
  - 3 drones circling at 1m high

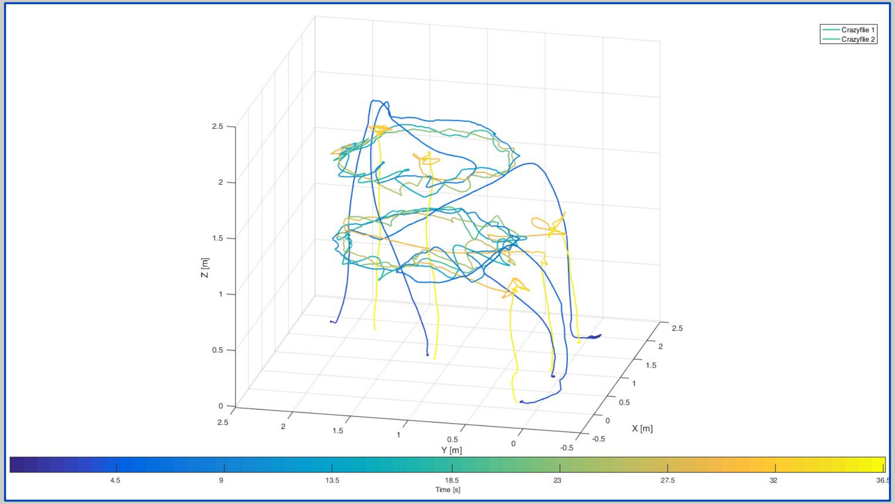


LPS Results - Multiple Crazyflies

LPS Results - Multiple Crazyflies

- Flew 5 Crazyflies in a multi-planar, parallel, concentric circular formation
  - 3 flying at 1m
  - 2 flying at 1.75m





Conclusions

Accomplishments:

- Successfully achieved circular flight from the simulations
- Successfully able to do simple formation control of multiple Crazyflies
- Discovered that the Kinect 360 was able to fly only 1 Crazyflie

Future Work:

- Continue development with Python API
- Read localization data from logs in real time using URIs
- Develop and tune higher level cooperative control algorithms

References

Peter Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB.* Springer, 2011. isbn: 9783642201431.

Zachary Dydek. *Adaptive Control of Quadrotor UAVs: A Design Trade Study With Flight Evaluations.* IEEE Transactions on Control Systems Technology, 2013. Vol. 21 No. 4.

Crazyflie Python Library. url:https://github.com/bitcraze/crazyflie-lib-python

Crazyflie Clients Python Library. url:https://github.com/bitcraze/crazyflie-clients-python

Bitcraze Wiki Site. url:https://wiki.bitcraze.io/

Q&A