Tyler Hall

CPE - Fall 2017

Real Time and High Fidelity Quadcopter Tracking System

Senior Project

In completion of requirements for Computer Engineering Department

California Polytechnic State University, San Luis Obispo

Author : Tyler Hall

Professor Advisor : Lynne Slivovsky

Abstract

This project was conceived as a desired to have an affordable, flexible and physically compact tracking system for high accuracy spatial and orientation tracking. Specifically, this implementation is focused on providing a low cost motion capture system for future research. It is a tool to enable the further creation of systems that would require the use of accurate placement of landing pads, payload acquires and delivery. This system will provide the quadcopter platform a coordinate system that can be used in addition to GPS.

Field research with quadcopter manufacturers, photographers, agriculture and research organizations were contact and interviewed for information on what components of a quadcopter system were lacking and what barriers currently limited desired drone operation. Distilling this information and after exploring various projects in the field of quadcopter and autonomous control, the idea was found to develop a system that could track the motion of quadcopters to jump start other projects.

Specifically, live feedback was explored to be used as hardware in the loop testing systems where commands are relayed to the quadcopter and its response can be accurately measured. This can be extremely beneficial in new equipment testing such as new propeller design, motor design, and frame response. A further stretch objective for this project is to unify input commands to the quadcopter with its physical position in order to train control systems to fly new platforms running “piloted” platforms such as BetaFlight, RaceFlight and KISS platforms typically associated with drone racing as well as hobby grade semi-autonomous flight controller such as ArduPilot Mega (APM) & PixHawk.

This was done as a senior research project at California Polytechnic State University - Computer Engineering department. Creator and designer Tyler Hall with Advising Professor Lynne Slivovsky.

Table Of Contents :

1 - Introduction

1.1 - Background

Quadcopter flight has been a recent phenomenon that has prompted new developments and research in machine learning and control systems. With immediate accessibility to the general public some of these research advancements can be tightly coupled with the needs of the industry. Those currently using quadcopter for photography, research, surveillance and other uses have need of developments that are currently limited by current technology. Because of a quadcopters simple fabrication and design, they are suitable for a range of flight platform needs including acrobatics, payload delivery and mathematical model simulations.

The proliferation of new quadcopter platforms has been made available to the general public with varying interests from professionals to researchers to avid hobbyists. With the cost of the platforms reaching into sub $100 with reasonable quality these platforms provide a new basis of research and exploration. With such competition and availability of new components, industry leaders, entrepreneurs, students and researchers are quickly looking for means of reducing cost of manufacture, improved capabilities and stability in flight.

This recent advancement has allowed students and entrepreneurs exciting exposure with limited means of funding to explore new ideas. There are key limiting factors in quadcopter research though namely motion capture systems to track and gather data about a quadcopters performance. Traditionally, new platforms have be tested against mathematical models and the live capture of the quadcopters flight via motion capture systems. One such example is the research being done by the “flying machine arena” at ETH Zurich Institute for Dynamic Systems and Control. With a 10x10 meter area in order to support “a high-precision motion capture system, a

wireless communication network, and custom software executing sophisticated algorithms for estimation and control." {1}

This provides a unique viewpoint into the operations and flight characteristics of the quadcopter. Variations in sensor readings, mechanical error, propeller design, mounting and balance as well as many other factors can affect the flight performance and attributes of a quadcopter. Additionally, if testing an autonomous self contained quadcopter system, an external tracking system has been traditionally used to verify and validate paths flown. This can in turn provide valuable data to correct or change parameters and variables within the autonomous system. {2}

1.2 - Initial Research

Preliminary research was first done by building a quadcopter for basic flight characteristics as well as flying four other platforms. Exploration was done to find what platforms were accessible and within reach of student expenses, flight characteristics and payload capacity. The first platform was designed in mind to carry a payload of at least 1.0 pounds, be modular to carry an APM 2.6, MavLink and Raspberry Pi for data logging and telemetry. The others were obtained through borrowing from friends, family, or personal purchase.

F450 Platform
Flight controller APM 2.6, MavLink for Live Serial Communication. Supports up to 1.0 pound test payload which includes the flight controller and raspberry pi. Flight time of Approximately 9-12 minutes. Including spare 3 batteries, Props and Extra frame.

Figure 1 - F450 platform

3DR Solo

Has the Pixhawk 2, MavLink, Digital Video downlink, expandable port for accessories.

Figure 2 - 3DR Solo

Bitcraze Crazyflie 1.0/2.0

Smallest of the drones. These are considered “nano” quadcopters with a payload of about 6 grams. This also includes live telemetry data over a 2.4 ghz Nordic NRF radio with companion USB for data capture.

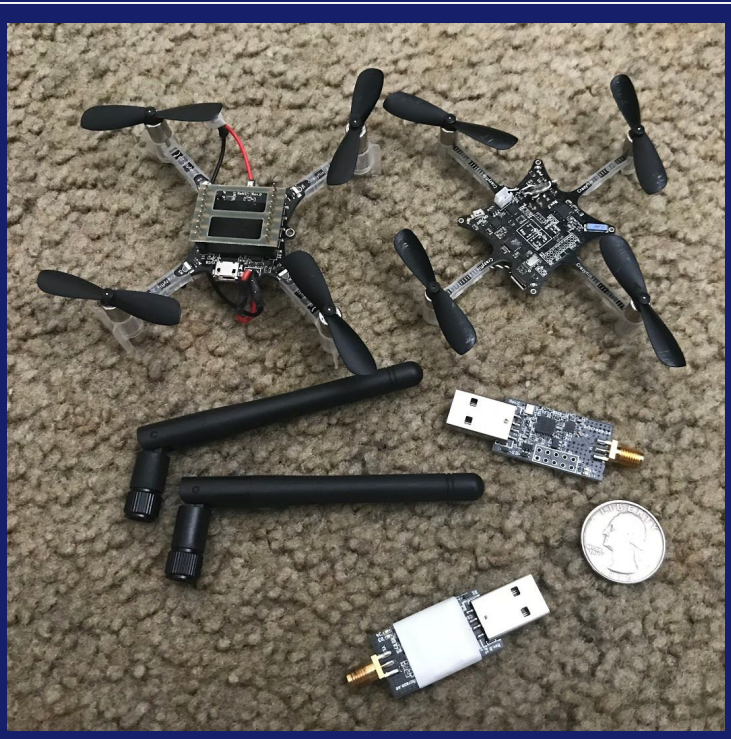

Figure 3 - “Nano” Quadcopter

Armattan Chameleon Race Quadcopter
5” racing quad with a F3 Betaflight Flight controller, 1300 mAh 100C battery, FrSky Taranis Radio,

Figure 4 - Chameleon

1.3 - Field Research

Field research was done by first contacting several quadcopter based companies and holding personal interviews. Many of those who interact with quadcopter based systems and services were interviewed to try and get a wide of picture as possible. The goal was to find pain points and limitations with the current setups, popular hardware platforms, typical software packages and commonly used technology. This also gave a interpretation of the current market and temperature for potential use cases as well as policy and regulation. These technologies that were

discovered were distilled down to what could be done in the span of a senior project. The focus was determined to allow for a platform that could track the position and orientation of quadcopter testing.

1.3.1 - Email Outreach

A short email was sent to approximately 50 companies found in the field of quadcopter use. These people ranged from researchers, to photographers to hobby store owners.

Subject :: Quadcopter Research

Hello!

My name is Tyler Hall, and I'm a computer engineering student at California Polytechnic State University (Cal Poly - SLO). I'm doing some research for my senior project on quadcopters and their potential use cases. Mostly, I'm trying to get a sense of how current companies workflow's match with launching, managing, and retrieving a drone including what equipment in use.

I'm currently working on a technology to manage the takeoff, landing and equipment management to provide a fully autonomous system.

I hope that you could take some time to respond and would like to perhaps do a short interview (10 - 15 min) about your current workflow, the struggles and success you have and where you see your company moving to in the sense of technology used.

Thanks for your time,

-Tyler Hall

thall09@calpoly.edu

1.3.2 - Question list, Cue Card For Interviews

.:: Value Propositions ::.	What do you deliver to the customer?
.:: Customer Segments ::.	Who are your main clients and customers? Size of those customers and business you do with them?
.:: Channels ::.	How do you find new customers?
.:: Customer relationships ::.	How do you keep that relationship with your customer? How do you include future business?
.:: Key Activities ::.	What is your key activities you must do to meet your customers needs? - Walk me through an experience where you're out on a job for a customer. - If selling drone packages Walk me through the kinds of features that you look to build for in your product. How does a customer use your product?

.:: Cost structure ::.	What key resources are most expensive?
.:: Policy ::.	Any certifications that you must have? Do have? Any certifications that the customer must have?
.:: Data & Software ::.	What kind of data do you collect? - How does this help your customer? - How does the customer have access to this data? - Software to process the data
.:: Future Connections ::.	If you're ever in central coast area could I come along?

1.3.3 - Outreach Response

The response was generally very positive by contacting these companies. Many spent their valuable time and providing insights to the drone industry that would not have been gathered any other way. The variety of questions gave a good ground for the interviewee to have open ended conversations and allowed the direction of the conversation to revolve around issues experienced with their current technology set.

Several recurring themes did happened between the three main groups of those interviewed. The main categories can be considered as photography, agriculture and sale of drones and parts. With all three of these groups, five key areas were found to affect all of them.

Five Core Results From Field Interviews

- Training Expenses (Part 107)
- Landing and Takeoff Safety
- Inconsistent and Confusing Laws and Policy (Government)
- Flight Time Limitations
- Accurate Flight Locations and No Fly Zones (NFZ)

1.3.4 - Field Research Conclusion

The results from the interviews provided a clearer context for the tracking system and provided additional parameters. In discussion with the interviewees, the idea of a trackable quadcopter system was presented and discussed. A majority of the interviewees indicated that if the system could land and takeoff accurately, the training factor could be significantly reduced. For example, if a drone could reliably land, switch batteries, and take off again with no human intervention then operation could be done by clients such as Vineyard Owners, Farmers and Police. On three separate occasions and without prompt, after being explained the tracking system, the interviewees indicated how a mobile setup for this tracking system that could be left with a client or the operator would provide ease of use and expand the customer base with minimal training time.

1.4 - Project Goals

1.4.1 - Design Requirements

The project is to create a Real Time and High Fidelity Quadcopter Tracking System under $1000 and be mountable on common quadcopter platforms. The system is designed for those in development such as students and researchers. As such, the design is to provide an interface to the live coordinate system via common protocol for consumption. This data should also be stored and used for consumption in matlab

or other mathematical programs and programming languages. The parts are designed to fit on common quadcopters so that the weight and size requirements of the tracking module(s) may fit on the quadcopter platform without adversely affecting flight performance by size and weight constraints. The cost must be reasonable and attainable for a senior project.

1.4.2 - Engineering Requirements

These engineering requirements are more explicitly laid out below based of the known end users and the design requirements.

1. Accurate quadcopter tracking within a +/- centimeter accuracy.
2. Live capture of coordinate system with minimal latency (Max 200ms)
3. Output the coordinate system via common protocol for other program consumption. UART, SPI, STD_OUT or some other common protocol.
4. Support for multiple tracked devices.
5. Must be a system that is small and mobile. Should not take more than 1 hour to completely setup, and takedown to move to another location.
6. Tracking must be consistent in a wide array of test environments, indoor, outdoor spaces must be trackable.
7. The cost for the tracking system total should be less than $1000 USD

2 - Initial Implementation

2.1 - Modified webcams and OpenCV

2.1.1 - System Design

The initial challenge was to find a working alternative to traditional motion capture systems. We know that we need to secure a 3d coordinate system and that this

system must be done cheaply and efficiently. Looking at traditional computer vision solution, the first idea was to use a bright object, or point to lock OpenCV and give tracking. This has been done many times before and is well documented by many sources. Websites such as www.learnopencv.com provided good resources to create tracking programs with OpenCV API for C++.

2.1.2 - Hardware Design

The assembly of the smaller 5" quad was chosen due to the size constraints of the flying area available at campus. The area is approximately 6x6 feet and is enclosed in the Computer Engineering department's "capstone" room. Assembly was relatively straightforward following the manufacturer's recommendations. Configuration via betaflight also was a fairly quick process with minimal issues.

Figure 5 - Chameleon building setup. Before assembly

For tracking, a bright paper with color (green) was attached to the Chameleon race quad and tracked using modified sample code set from www.learnopencv.com.{3} A color key was then selected for the color to be tracked and all outside areas had similar green color removed. This was a first attempt, and was easy to quickly setup. A video recording of the quadcopter moving was taken via

laptop webcam. Although incredibly cheap, quick and easy to setup the cons of the tracking system was immediately apparent as accuracy and reliability proved difficult with this simplistic approach. The This proved to be very inconsistent as the drone would move too quickly and the tracking library would lose tracking of the paper if the the quadcopter change orientation too much.

Figure 6 - Chameleon with target color for pixel color target tracking

2.1.3 - Pixy (CMUcam5)

As a final camera test, the popular and open source Pixy camera was used to do a similar example of the color tracking. There was no expectation for this to perform any differently than the color tracker experiment before since it was operating of the same basics of a single video camera tracking a color object. It was in a toolbox of recent projects for other projects. Similar results were obtained. {4}

2.2 - OpenCV and IR LED's

In an effort to limit tracking loss and also to provide higher contrast, a potential solution was suggested by a co-worker. The idea was to increase the contrast of the viewed RGB video feed by removing the IR filter from a common webcam then place IR LED's on the quadcopter to be the "tracked object". The components were assembled by carefully deconstructing a Logitech webcam and removing the IR pass filter. This allows the webcam to view IR light directly and see with higher contrast IR light sources. IR LED's were connected to the quadcopter flight controllers 5v VCC to provide power.

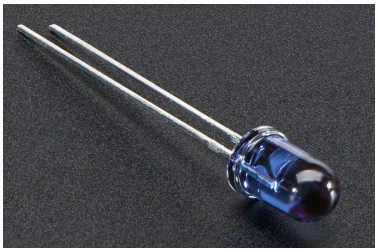

Figure 7 - LED Blaster Used in testing

Figure 9 - Modified Webcam with IR filter removed

This gave a much higher contrast and using essentially the same code as the color tracker, was able to get better performance with orientation change and

movement speeds. Although this was better, there was some serious refractions of outside IR (light from outside) that rendered this approach as unusable.

2.3 - Initial implementation Conclusion

The system of approach to tracking was naive and was bound to fail to meet the desired requirements of the system. Although the implementation was insufficient it did provide good context into requirements of what should be approached next. For example, the mono view of a single camera can not provide sufficient information about depth tracking unless there is a very well defined object that can be tracked. Object tracking and detection has its limits. A second camera tracking could in theory be used to secure a second view and establish that depth distance. This was starting to be explored when stumbling upon a better solution of using the HTC vive and Valve VR as a prototype tracking system.

3 - Second Implementation

3.1 - HTC Vive Virtual Reality Tracking

Figure 10 - HTC Vive and Triad Semiconductor unboxing

3.1.1 - Valve Lighthouse background and Tracking Design

While searching for an alternative to the mono camera implementation it was discovered almost by accident about the HTC vive. The Vive is a Virtual Reality (VR) gaming system where users wear a headset, and use controllers to play video games in an immersive simulated environment. The interesting part is the technology required to track the headset and the controllers accurately and quickly for seamless game play. The concepts behind the Valve VR tracking system are a great fit for a motion capture system.

The problem of accurate spatial tracking is not an issue isolated to any one use case but is a common problem. Valve decided to attack the problem in an interesting way. Essentially the system works as follows:

1. Two "lighthouse" boxes are placed in the corners of the space to be tracked.
2. These lighthouses emit sweeping "planes" of laser light at a very specific rotational speed of 60 rev/sec. First scanning in one direction, lets say X. This is highly accurate by custom bearings and material thickness accurate to a micron.

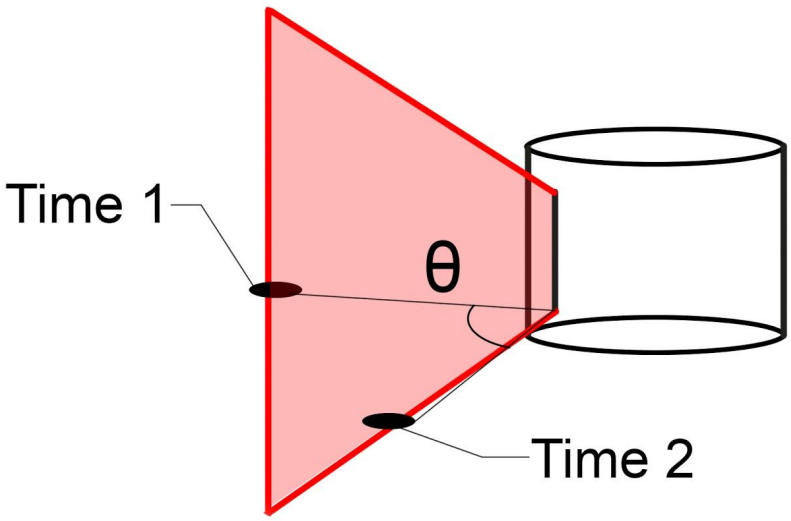

Figure 11- Diagram to demonstrate signal sweep sequence

3. The time is measured by the IR sensors, and with the distance between the sensors known, simple trigonometry is used to to established distance between the sensors on the controller and the base station. {5}
4. This is done a second time in the other direction, Y, to determine the second plane.

5. A second box is in the room, which also scans in its X, Y. This secures the third coordinate.
6. The more sensors that are able to see the light houses, the more calculations can be made and the more accurate the position
7. IMU data is also used to estimate position based of the last known scan to interpolate position between laser sweeps. IMU drift is corrected when the {6}

More mathematical analyses can be found Trammell Hudson's projects {7} where some of the translational math is explained in depth as reverse engineered at the early stages of the HTC vive lighthouse release. At the time, I was working on understanding the mathematics behind this process and working with various Electrical Engineering and Mechanical Engineering Professors on understanding the linear algebra behind the fixed distance calculations. I was searching to find IR sensors that would match the bandpass filter spec needed for the Valve IR broadcast. To my luck, Valve had during the time working the project released a public hardware developer kit (HDK) for use of creating custom trackable controllers.

3.1.2 - HTC Vive System Setup

The valve hardware was set up in the corners of the capstone room in an appropriate space for the small 5" propeller sized quadcopter. A VR capable computer was secured with a suitable video card to support the minimum Vive requirements. All components were assembled by manufacturer's instructions with accompanying tools and software. The play area was configured and properly calibrated using the Valve VR tools for room size calibration.

Figure 12 - HTC Vive setup with Valkyrie Tracking program

Figure 13 - Virtual Reality computer

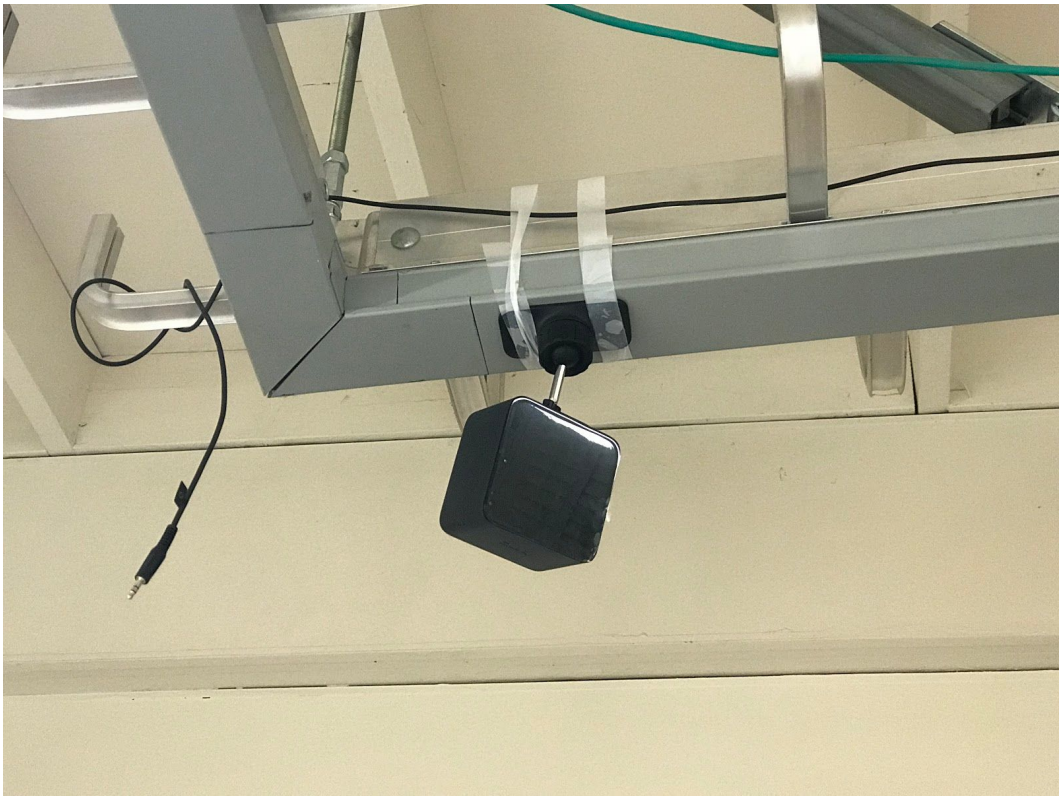

Figure 14 - Lighthouse placement on the enet cable scaffolding in capstone room

3.1.3 - Unity 3d Setup and VRTK

Development for the Valve VR is supported by the Unity 3d Development platform. This developer environment supports both a javascript implementation and more popular a C# program. Several small example programs were made to figure out the cross development with Valve VR. To explore how the game objects interacted with C# scripts the program "Hello Tracking" was created to have a single sphere follow a predetermined path and export this path to console for later processing by matlab.

3.1.4 - HTC Vive Controller Tracking

The interface between the HTC Vive and the entire tracking system including head tracking, camera view, and controller positioning was tackled by creating a unity project with several C# scripts. The HDK and SDK for the custom controllers register the new controller just as the standard controller is registered.

A scene is set up for the HTC Vive and the proper libraries are initialized to import tracking. Several VRTK objects are also imported for the headset and each of the controls as well as calibration settings for the floor and boundaries for the play area that was initialized for the HTC vive during setup.

Figure 15 - Before the custom controller was built, the vive controllers were test mounted and flown with the program created "Valkyrie"

The created program called **"Valkyrie tracking"** does several things to ensure tracking, data logging, and data visualization in the 3D space.

1. The Game environment is configured and connection is made to the Valve VR SDK
2. Game elements from VRTK are configured, imported and matched with user defined preferences (See screenshots) where materials can be applied by users for specific colors.
3. Vectors are bound to each controller via the ValveVR_Tracking.cs script. This script is heavily modified to not only include live tracking but to log data, build a visual path for the user to "see" in virtual reality the path of the flight, and to clear the path with a button press and output to a timestamped CSV log file for future processing in matlab.
4. Some effort was made to also read in this log file, and build the path again. The updated of unity has currently caused issues with this. A better and more standard means of object serialization and deserialization needs to be found.

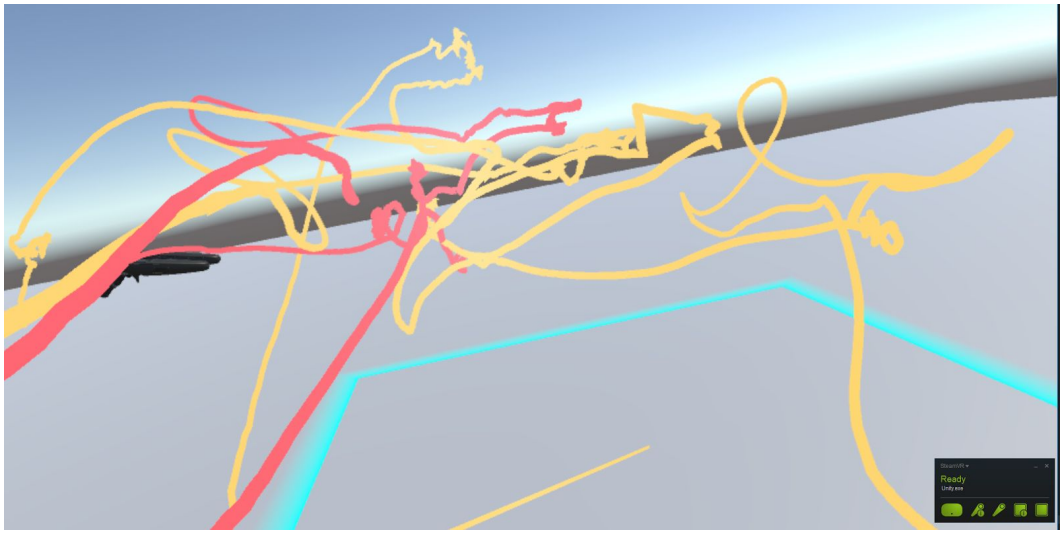

Figure 16 - Screen Capture of path tracking of two controllers.

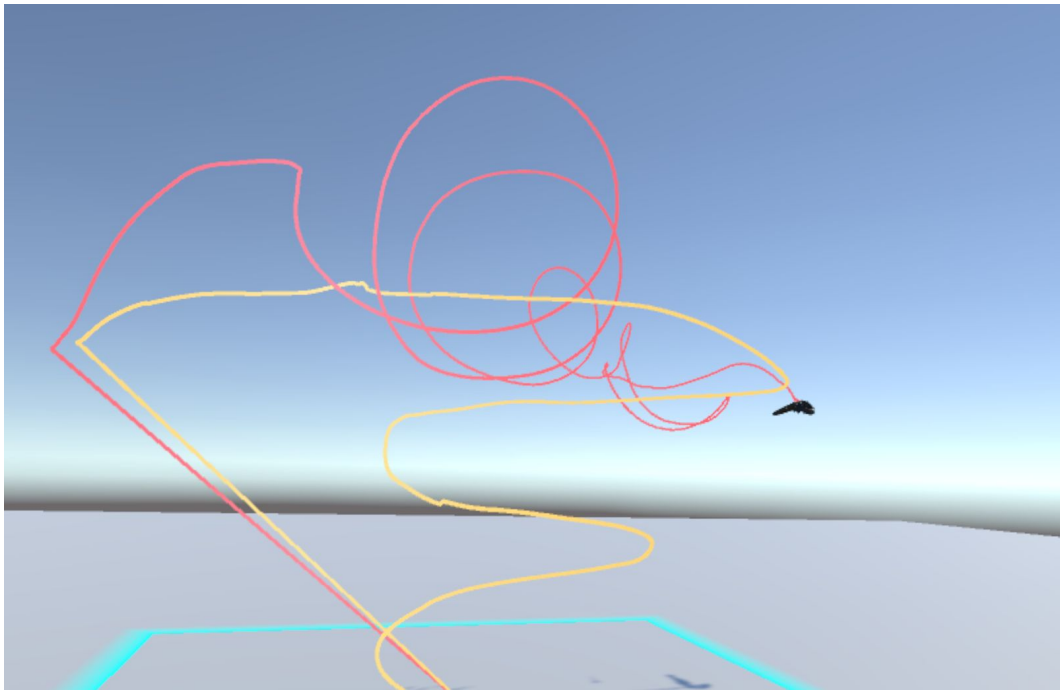

Figure 17- Another example and screen capture of the tracking system paths. Spiral and flip. (Both paths done with the Chameleon quadcopter)

The interesting component of this setup is that we now have a completely digital world that is interacting with the physical. Much like video games are played and interacted with by humans in virtual reality, a quadcopter can fly appropriate paths, interact with virtual objects, run machine learning algorithms to "train" a quadcopter to fly and accomplish objectives. This can all be done in simulation of course, but this gives us a new opportunity to test the hardware in loop with physical constraints and real world errors.

For example, in this experiment the quadcopter was flown through a "virtual hoop". Collision detection within Unity was used to track when there was a collision of the controller and any part of the virtual object. There is much to be explored with physical interaction in a virtual world. This could lead to further testing scenarios created by researchers and students that would otherwise be unreasonable to create physically, cost prohibitive or potentially unsafe. In this environment it is OK if the drone "crashes" into the wall, since the wall is a virtual one.

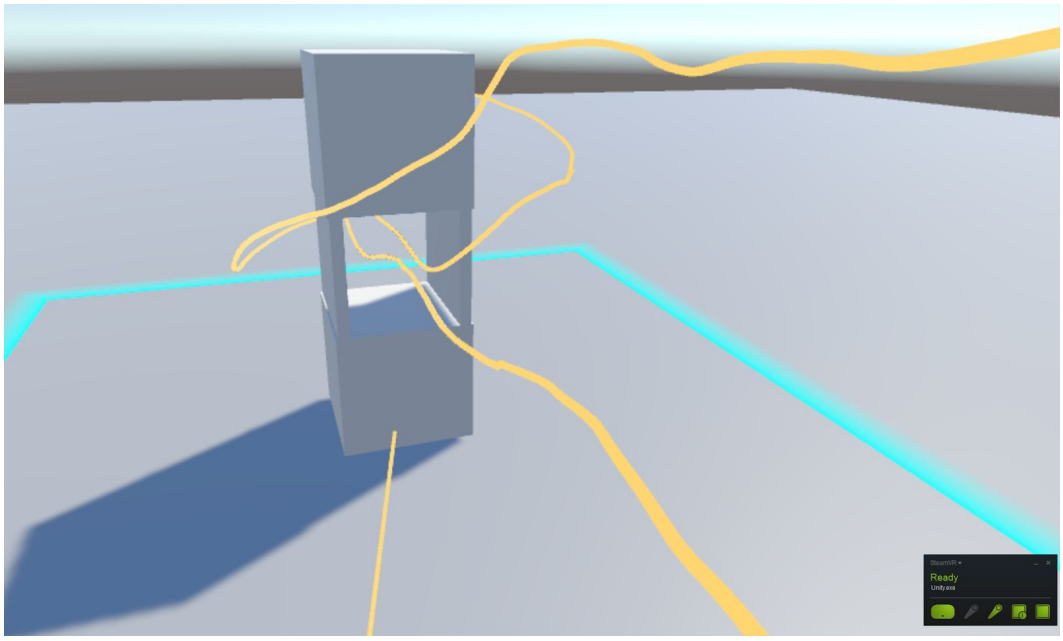

Figure 18 - Flying through virtual objects

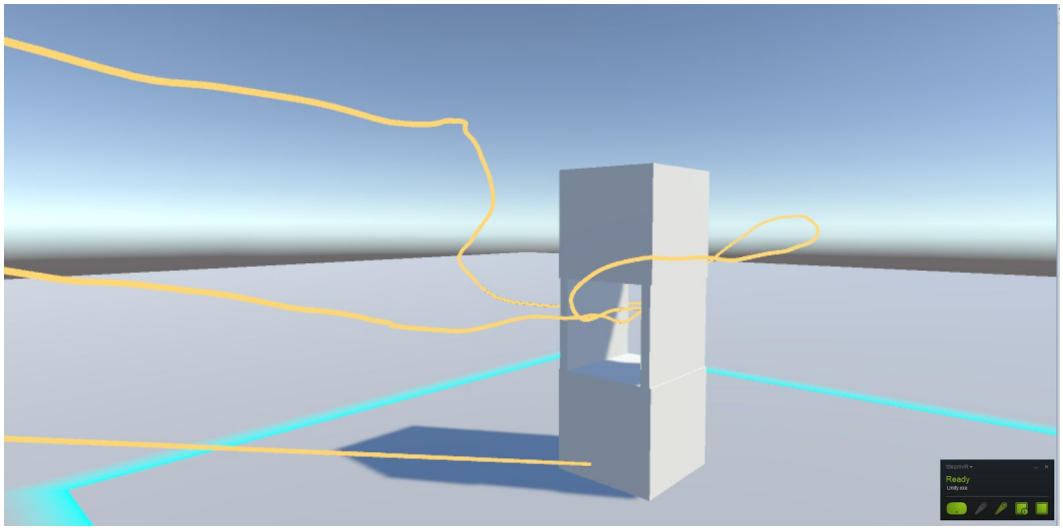

Figure 19 - Same flight path, different angle to show 3d path.

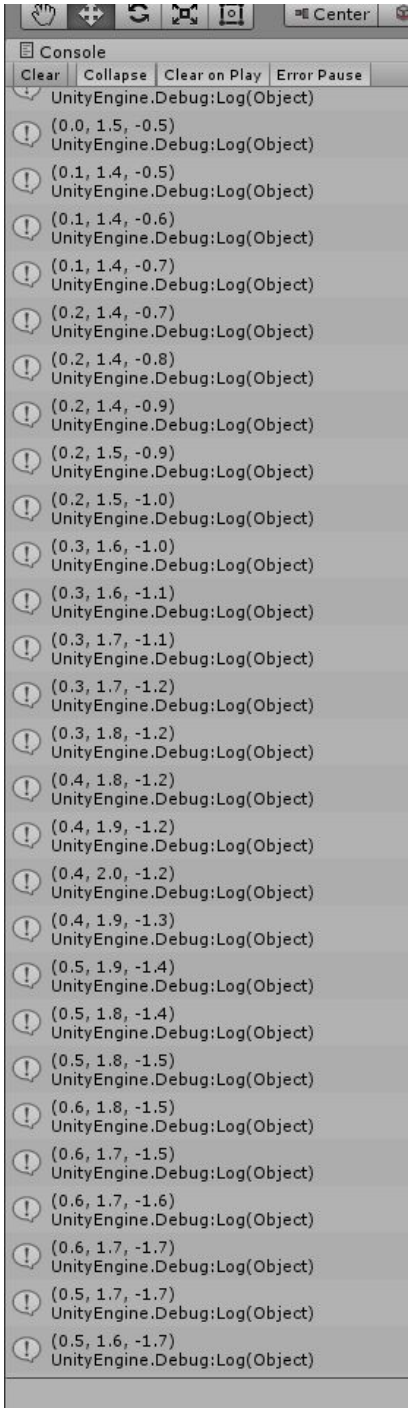

Figure 20 - Example log output

0.1893972,0,0.5,0
0.2024851,-2.543567,1.309017,-0.4755671
0.2089335,-2.543348,1.309039,-0.4756011
0.2352682,-2.543448,1.309019,-0.4757959
0.2521012,-2.543375,1.308833,-0.475844
0.2583549,-2.543381,1.308813,-0.4758636
0.2633626,-2.543315,1.308641,-0.4759456
0.274446,-2.54371,1.308698,-0.4760565
0.286701,-2.543726,1.308723,-0.4760863
0.2964429,-2.54375,1.308799,-0.4761227
0.4805347,-2.543849,1.308851,-0.476107
0.4872322,-2.543778,1.308881,-0.4760169
0.4982802,-2.543778,1.308964,-0.4759986
0.5127289,-2.543642,1.308968,-0.4760188
0.5196319,-2.543788,1.308827,-0.4760169
0.5309767,-2.543795,1.308835,-0.4760202
0.5422181,-2.543879,1.308793,-0.4759722
0.5531676,-2.543936,1.308751,-0.4760796
0.5644228,-2.543953,1.308747,-0.4760724
0.5760761,-2.543881,1.308642,-0.4760584
0.5867604,-2.543938,1.308627,-0.4761528
0.5979434,-2.543979,1.308598,-0.4761494
0.609076,-2.544078,1.308634,-0.4762172
0.6202853,-2.543859,1.308729,-0.4758036
.
.
.
.

Part of a .CSV output file generated by the system. <seconds, x, y, z> in meters
This can then be imported to matlab or consumed by other programing means for further analysis.

3.2 - ValveVR HDK

3.2.1 - Components and system overview

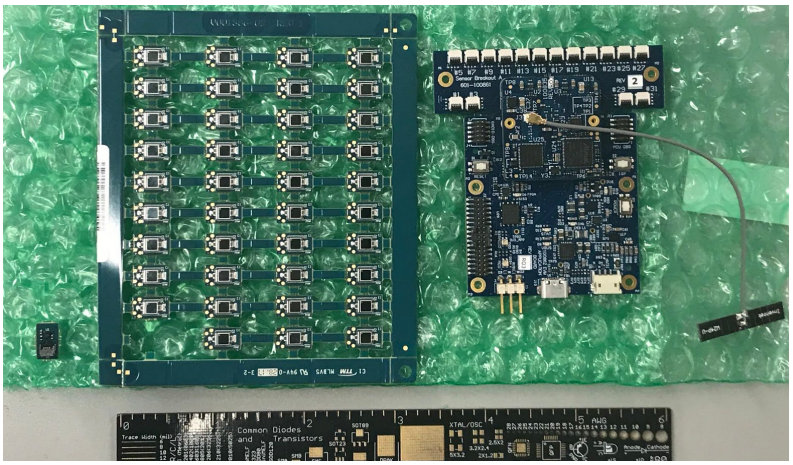

Figure 21 & 22 - Hardware Development Kit for custom tracker build

The valve HDK is directly purchasable from the manufacturer {8} from Triad Semiconductor. The tracking system comes with a licence guaranteeing that development and full scale production can be done with Valve VR without paying royalties. {9} The total kit comes with all the electrical hardware needed to make a fully custom controller trackable by ValveVR. Each component has a specific purpose and is described below:

- Watchman Core Module iCE40
    - This is the "core" module holding the ARM processor, MCU, and 2.4 GHz radio
- EVM Application Board
    - The "companion" board. Breaks out all readable lines and debugging and provides interfacing for calibration and PIN support
- "Chiclet" Sensor Modules
    - These are the IR sensors that are detecting the laser sweeps
- Sensor Breakout Board
    - The white tabs accept the Flex Cables to connect the chicklet sensors.
- Steam Wireless Dongle
    - Enables live telemetry, OTA updating and communication from the 2.4 GHz onboard radio to USB/COM
- Four packs of 8 4in Flex Cables (32 cables total)
    - Used to connect Chicklet Sensors
- 2.4 GHz Antenna with u.FL Cable

3.2.2 - Command line Tools and Calibration

Along with the HDK installation through steam, additional tools and configuration setups are given to load information onto the development boards. This

includes configuration files, initial connection to the system and other setup. This is necessary to get the boards configuration information before mapping any designed enclosures for sensors.

3.2.3 - Point configuration and Auto Sensor Mapping

It is at this point that we can discuss the what is entailed in designing and creating a custom enclosure. It is important to understand that in order for tracking to work, we must build an accurate model and match the placement of sensors to the model. The models used were created in solidworks 2016 where other programs that output .STL models would also be sufficient. This program was used by suggestion of the HDK. This model can be then used to specify sensor placement. A JSON file with the specified points, normals, and sensor number mapping needs to be created that will be uploaded to the board. This can be done one of three ways with the HDK.

Option 1: Manual placement of the points by hand calculation of the points and their normals. This is a tedious process that requires inspection of points in matlab and using formula in figure <sources needed> to compute the needed normal. This is not recommended as it is easy to make a miscalculation.

$$x_{normal} = \frac{x_{distal} - x_{surf}}{\sqrt{(x_{distal} - x_{surf})^2 + (y_{distal} - y_{surf})^2 + (z_{distal} - z_{surf})^2}} \quad (1)$$

$$y_{normal} = \frac{y_{distal} - y_{surf}}{\sqrt{(x_{distal} - x_{surf})^2 + (y_{distal} - y_{surf})^2 + (z_{distal} - z_{surf})^2}} \quad (2)$$

$$z_{normal} = \frac{z_{distal} - z_{surf}}{\sqrt{(x_{distal} - x_{surf})^2 + (y_{distal} - y_{surf})^2 + (z_{distal} - z_{surf})^2}} \quad (3)$$

Figure 23 - Formula for manual point/normal computation

Option 2: Automatic generation of sensor placement given the .STL model. In order to do this, an additional model needs to be created to be used as a mask to the

original model. Any area from the original model that is NOT covered by the mask will be included in a sensor placement test sequence. Several parameters are specified such as number of sensors and number of iterations to complete. Essentially the tool tries several combinations of sensor placements and returns the best fitting set.

Option 3: Manual placement of points, and normal lines in SolidWorks, and using the provided script to extract and pack the information into JSON format. This is efficient once the procedure is followed with strict adherence in SolidWorks.

3.2.4 - Geometry mapping and Sensor Placement Optimization

As a test. First simple objects were placed in the optimization engine to understand the "rules" and what geometric changes would do to the sensor placement. Simple shapes such as spheres, cubes and other initial design shapes were explored to see their characteristics.

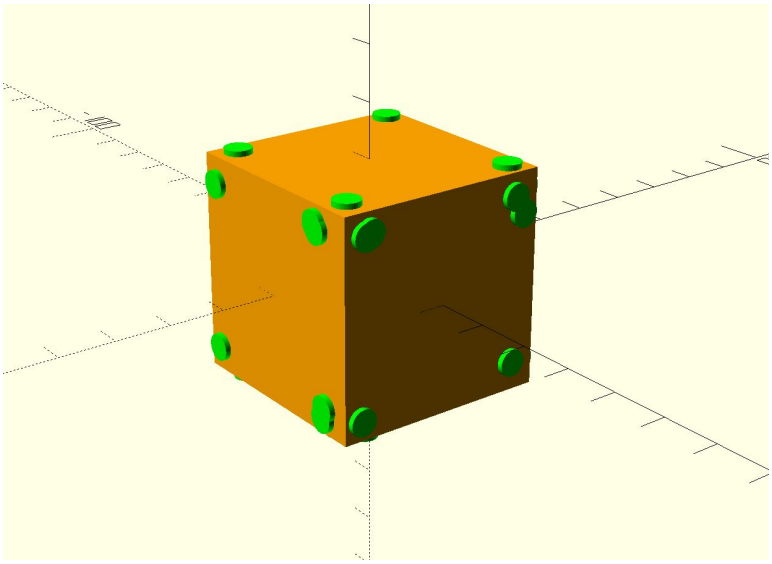

Figure 24 - OpenSCAD model showing generated sensor placement points

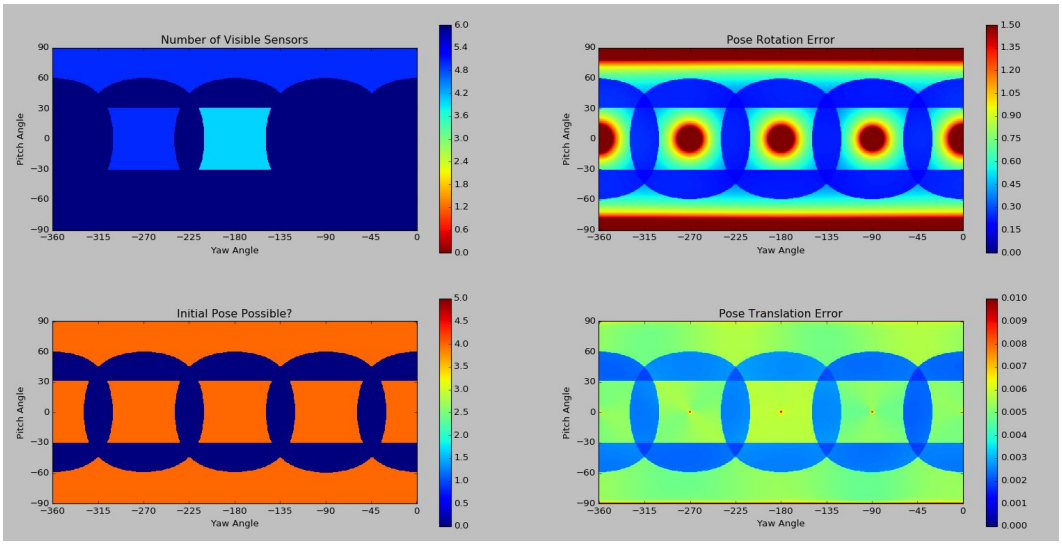

Figure 25- This is a 2D representation of metrics.

The 2d representation of metrics shows a few crucial pieces of information concerning the performance of the geometry and the sensor placements. The of Number of visible Sensors, Pose rotation error (angular position/orientation), Initial Pose possible (defines the positions required to be seen by the sensors before IMU interpolation initializes) and Pose translation error. For all these metrics blue is "good" and red is "bad". The gradient to the right shows the steps in between.

3.3 - Prototype Enclosure Creation

3.3.1 Initial Test Print - Sensor Placement with 3d printed parts

The initial test print was done using the "UFO" included example from the HDK training documents. This model was already include with the JSON file and was encouraged to be the "first try" for those starting out with the tracking system. This was 3d printed on campus at the dFab lab (digital fabrication).

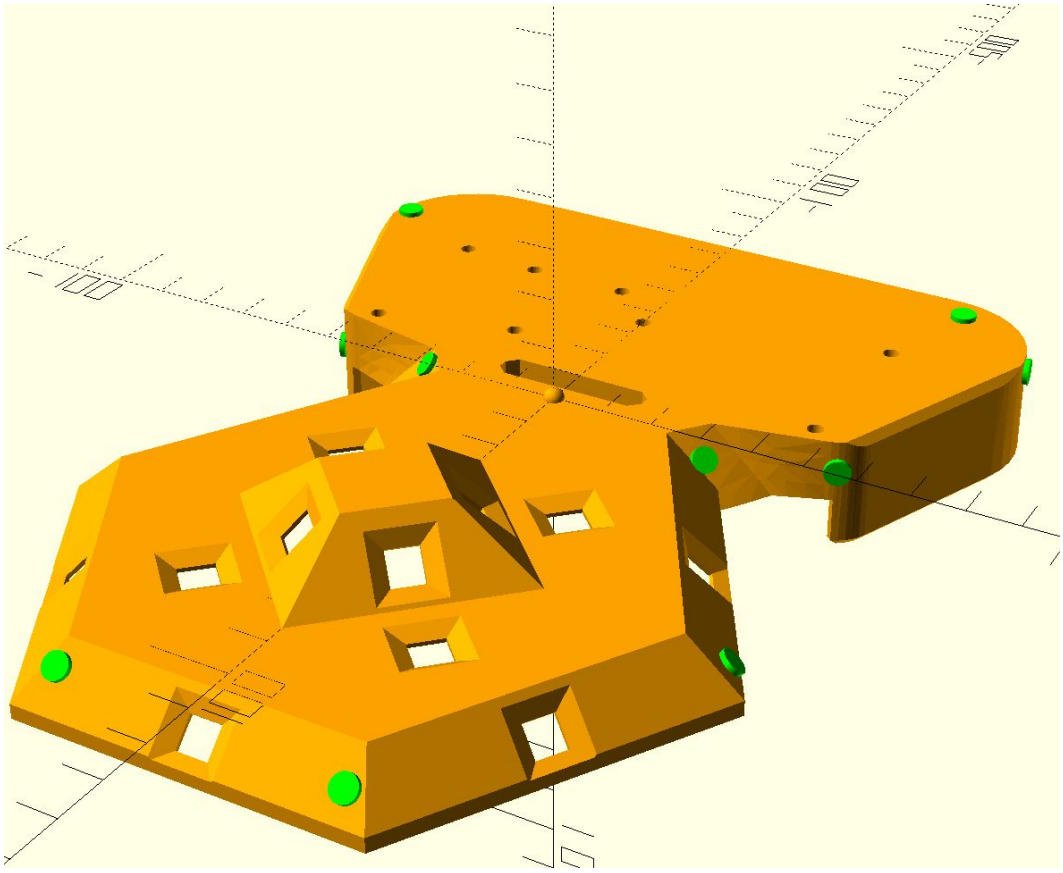

Figure 26 - Generated sensor placement on the UFO object.

The model that came out of this experience was not what was expected. 3d printing although very convenient is not terribly accurate especially if the machines are not sufficiently capable or calibrated. This initial print gave many considerations for future designs with the constraint of 3d printing. This part was not sufficient to use as the entire part was scaled down by approximately 97% of the size that it should have been. This caused the sensors to misalign in the ports and also to not calibrate correctly when running the HDK calibration for sensor alignment and IMU initialization.

3.3.2 Second Attempt at 3d Printing

The second test was the same UFO file but 3d printed. This was done with a calibrated machine. The issue on this was the sensor placements provided by the JSON

file in the HDK was malformated for the current version of the tools. So after the print, a few things were confirmed to physically fit such as the development board and sensors in sensor windows.

3.3.3 Third attempt with scratch build

Finally a custom build was started from scratch in SolidWorks. After several minor version iterations a final shape was found. This shape was based of the UFO example and was measured to fit the Chameleon quadcopter.

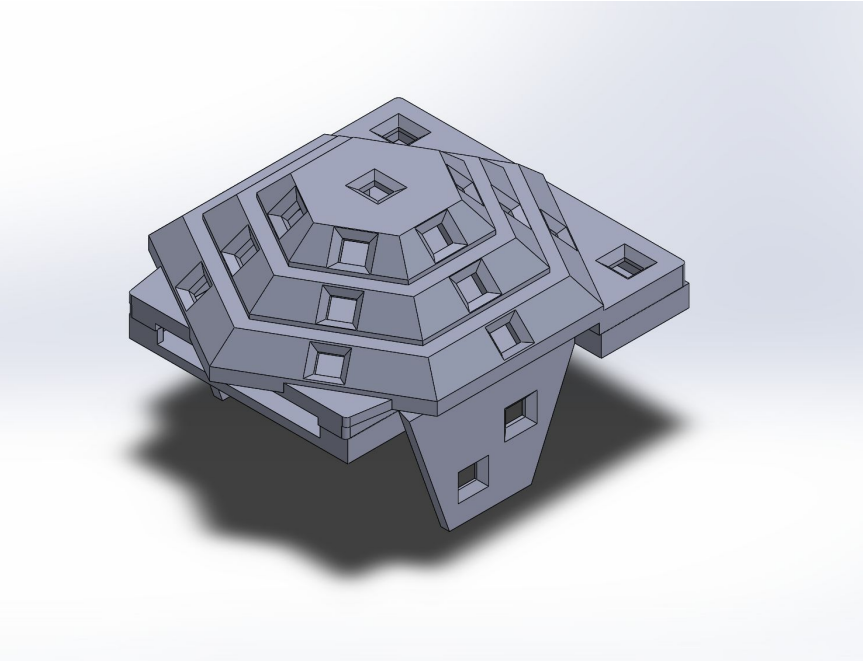

Figure 27 - Final Implementation For 3d Print with windows

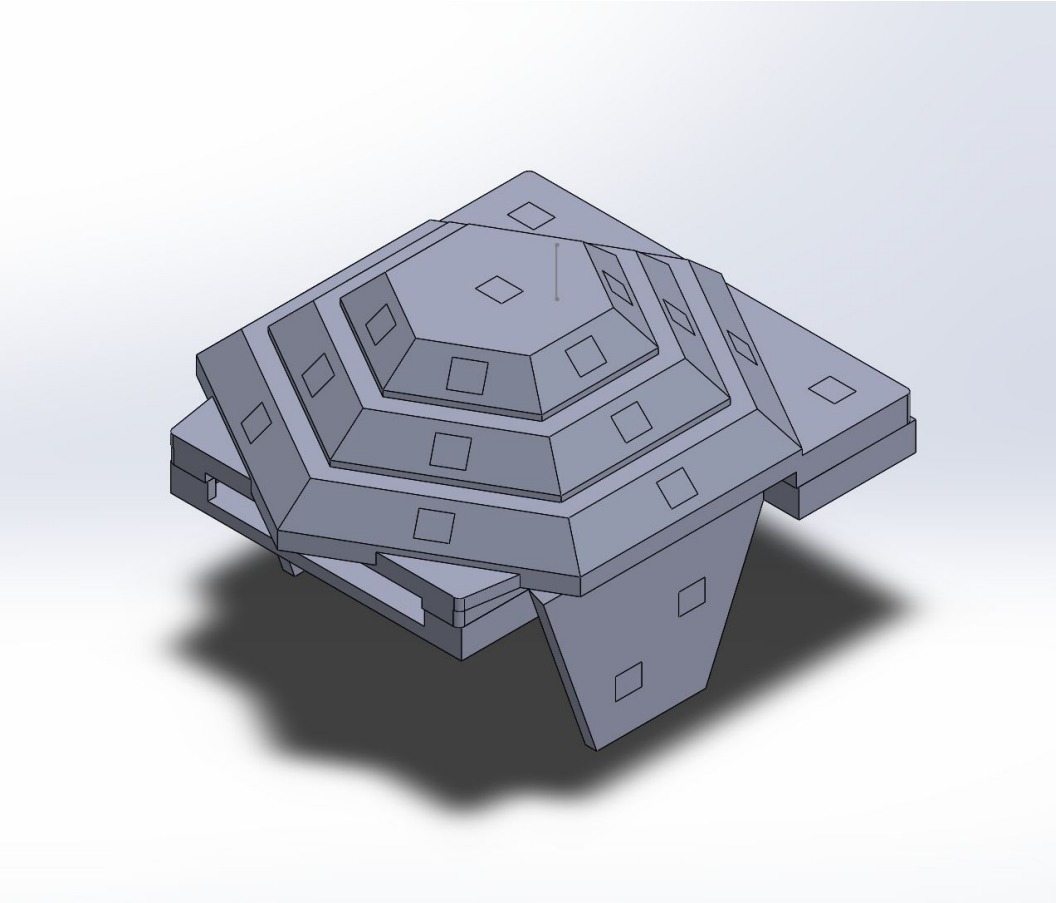

Figure 28 - Final Implementation For 3d Print without windows for surface normal calculation

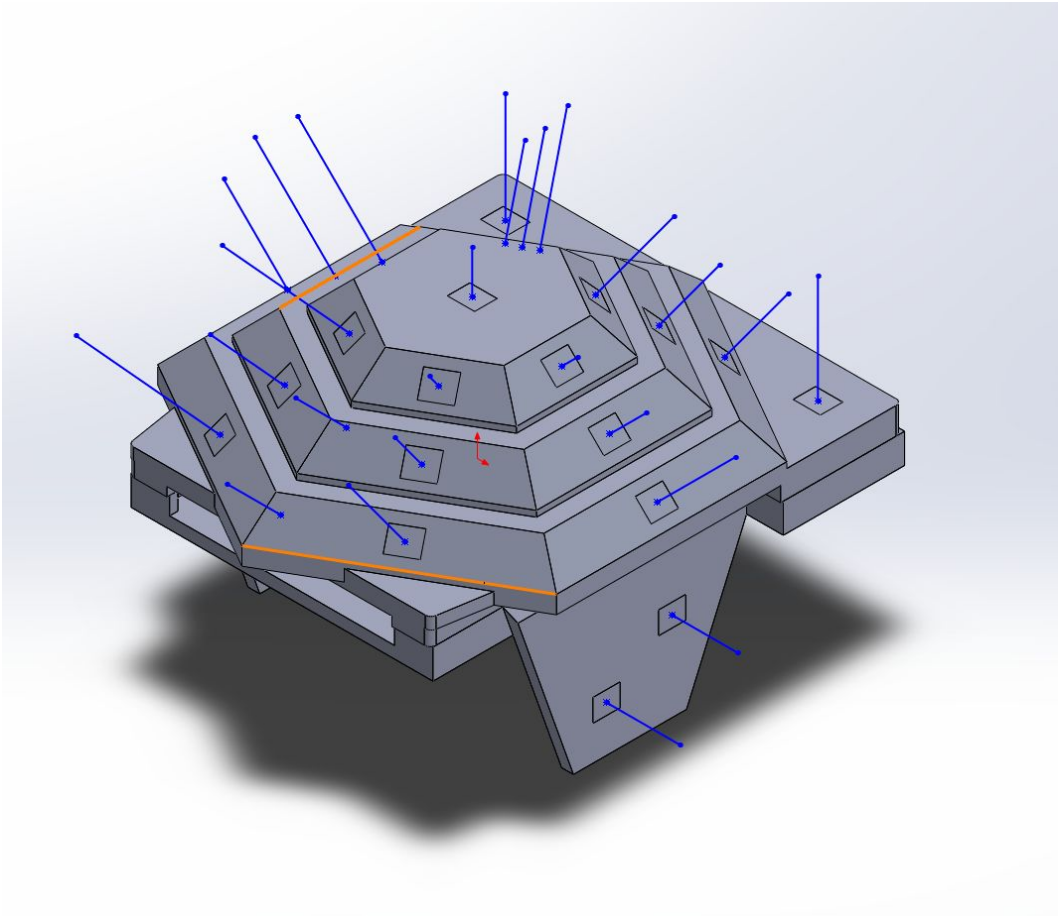

Figure 29 - Showing the point/normal setup for script calculation for JSON configuration

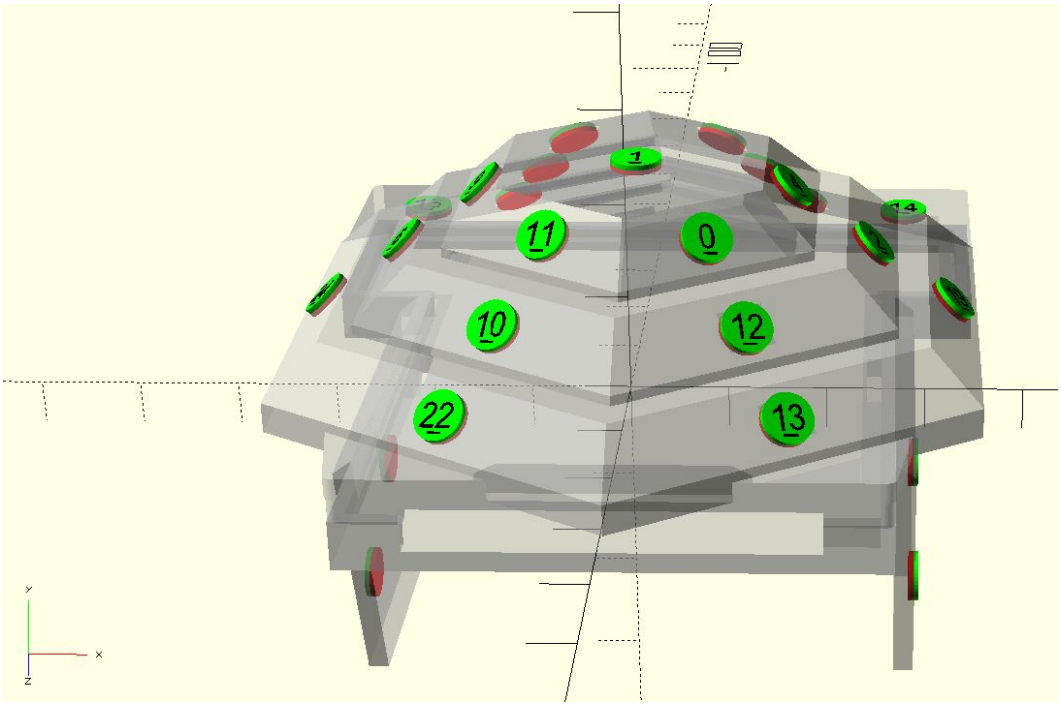

Figure 30 - Sensor assignment after model generation

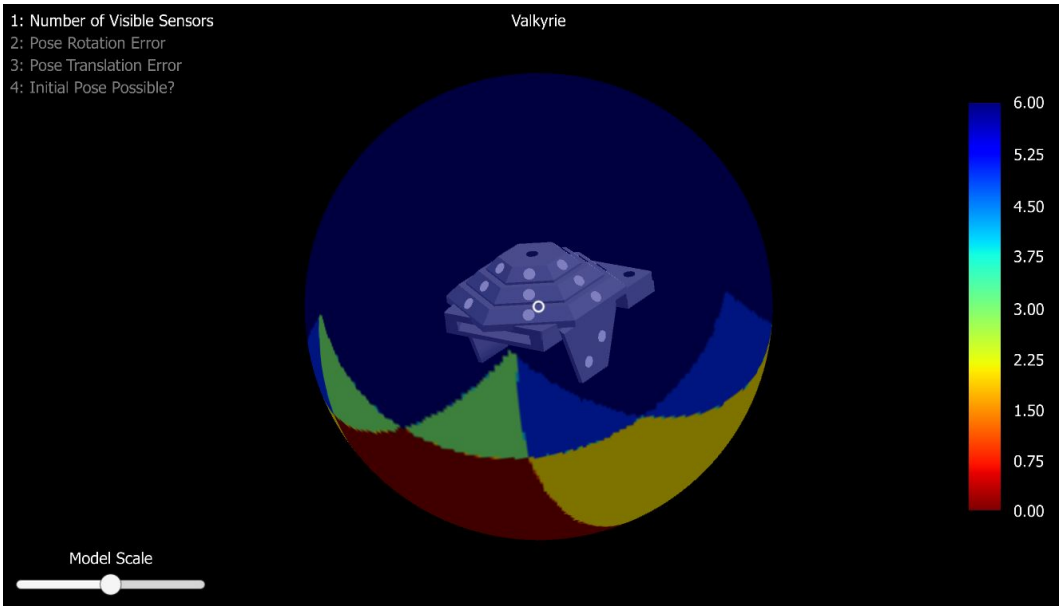

Figure 31 - 3d Representation of the design metrics

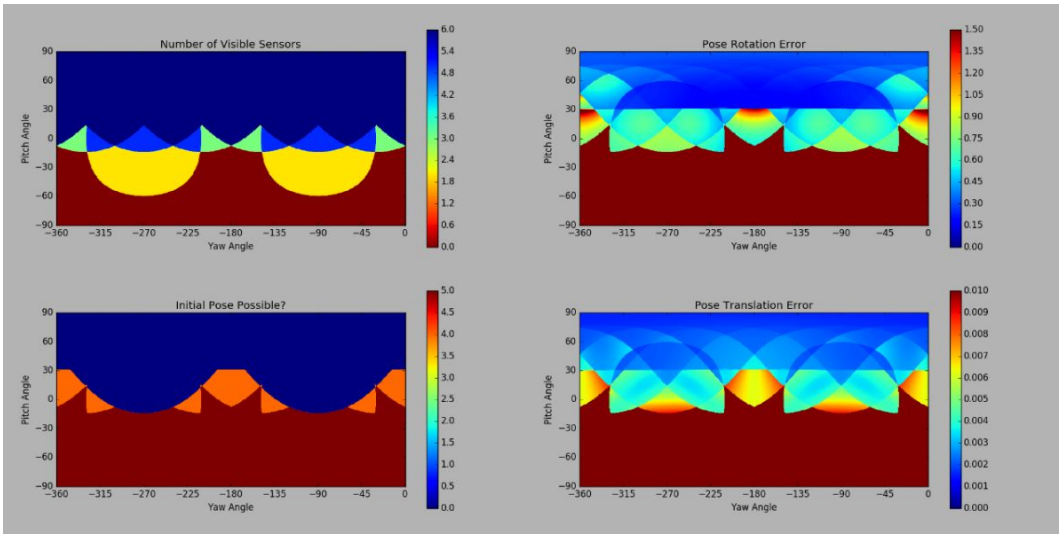

Figure 32 - 2D representation of the design metrics

As seen from the analysis above, the potential viewed sensors is essentially on the top half of the visible sphere with the exception of the side wings. The additional small wings were essential to get more space for initial pose possibilities. In the HDK documentation they suggest having at least 75% of the space a shade of blue for proper device initialization in the tracked space.

This model was then 3d printed using the calibrated 3d printing machines in teh EE technical department. Working with them, we were able to produce a High quality print in which the sensors fit perfectly in the windows. Although the accuracy was now configured and the sensors were in place, some other errors were made such as allowing for space in the enclosure for pin placement, various ports and overhangs. Unfortunately this caused the calibration to fail. At this point in time I assume that the IMU position as specified in my model did not fit within the error margin for where it was placed in relation to the sensors.

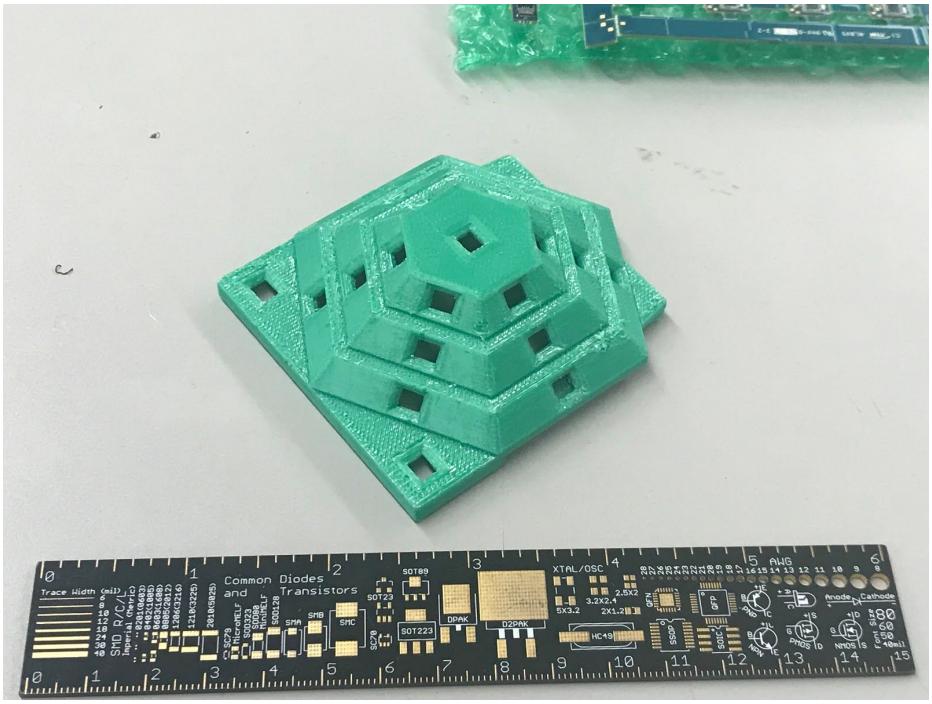

Figure 33 - Top shell of the 3d print

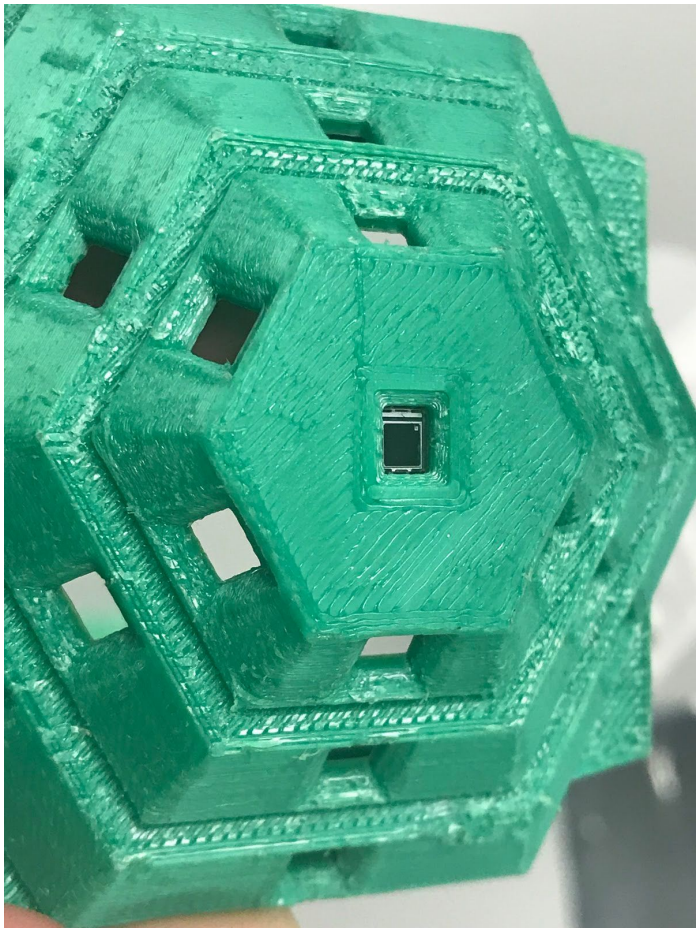

Figure 34 - Sensor placement and configuration

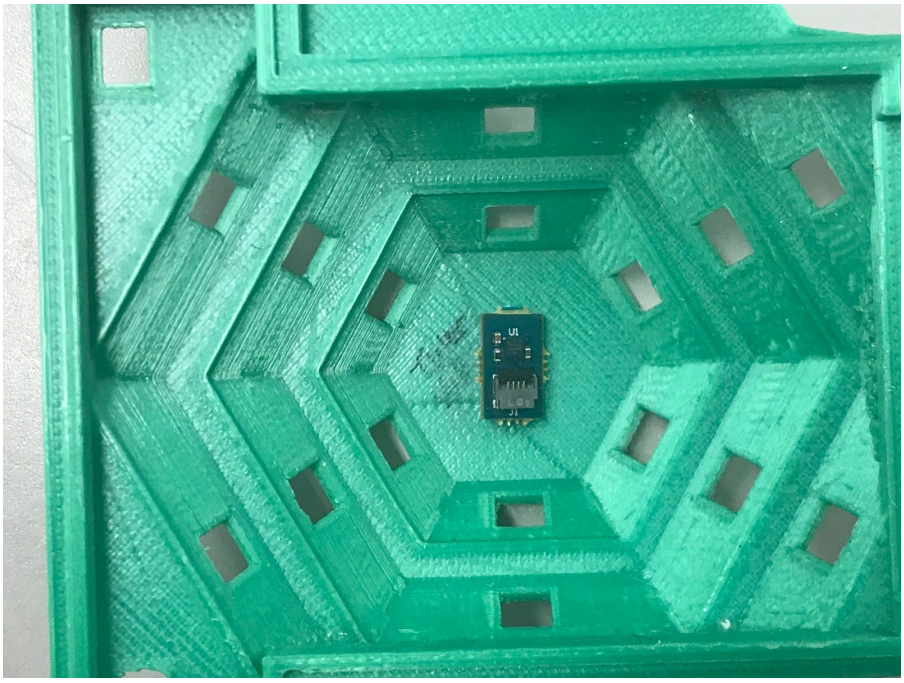

Figure 35 - Inside of the shell. Placement of the Chicklet sensor.

3.3.4 Enclosure Conclusion

The small mistakes made on the the third print could be overcome with small modifications to the model and reprinting. Unfortunately due to time constraints and printing time, I was not able to reprint to get a model that would calibrate with the HDK tools. This will be completed after the quarter and commented on a later date. This would have provided a completely custom enclosure for the tracking system allowing the quad to fly much more consistently vs. the controller strapped on the top plate. The tools do seem to be responded to the model correctly. I believe with one more print and small modifications we'd have a fully working tracking system. This would be a drop in replacement with the standard HTC vive controllers and would function with the "Valkyrie" program in Unity.

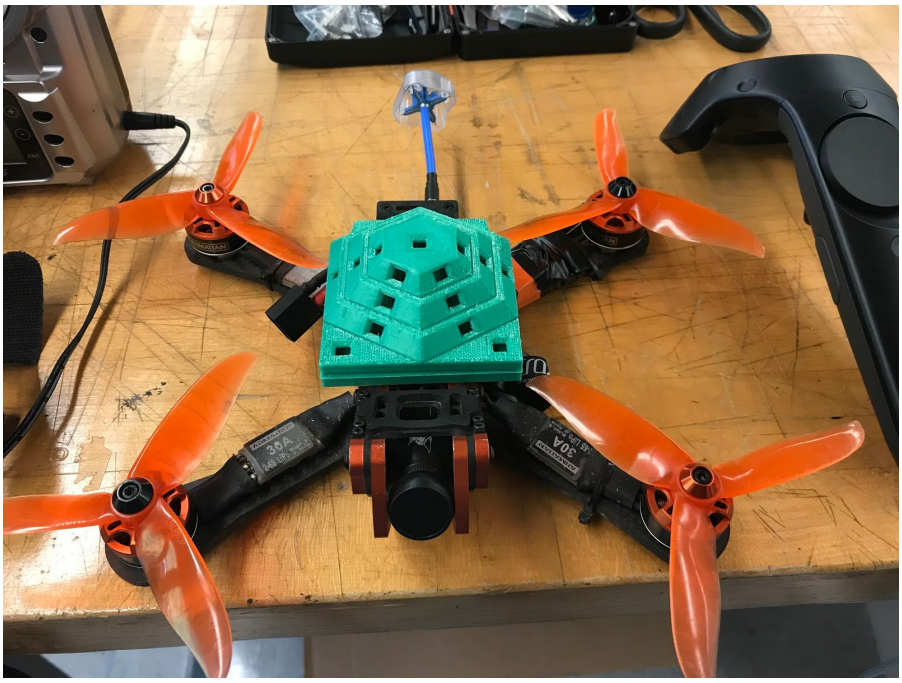

Figure 36 - Final Mounted Shell on Chameleon (front)

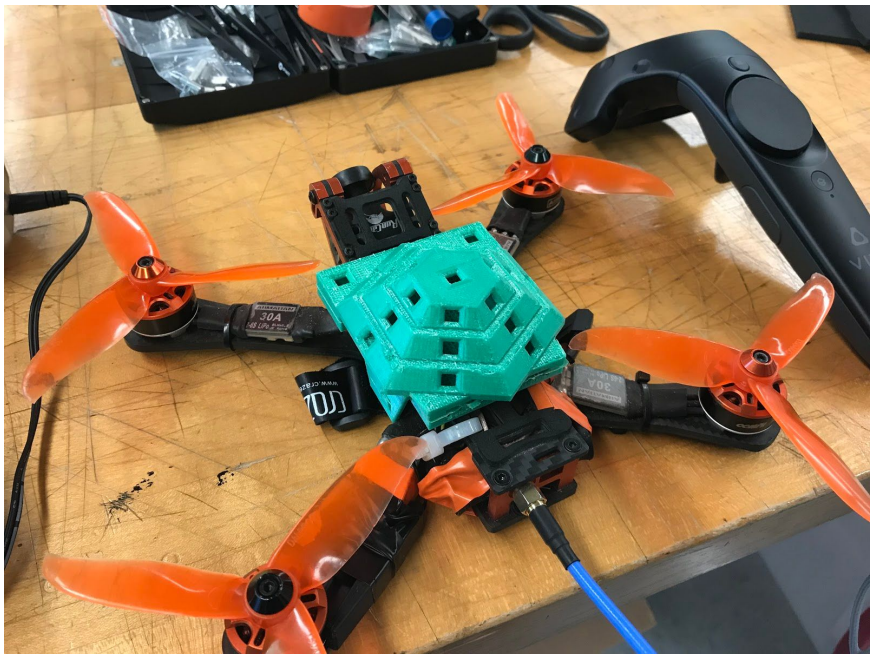

Figure 37 - Final Mounted Shell on Chameleon (back)

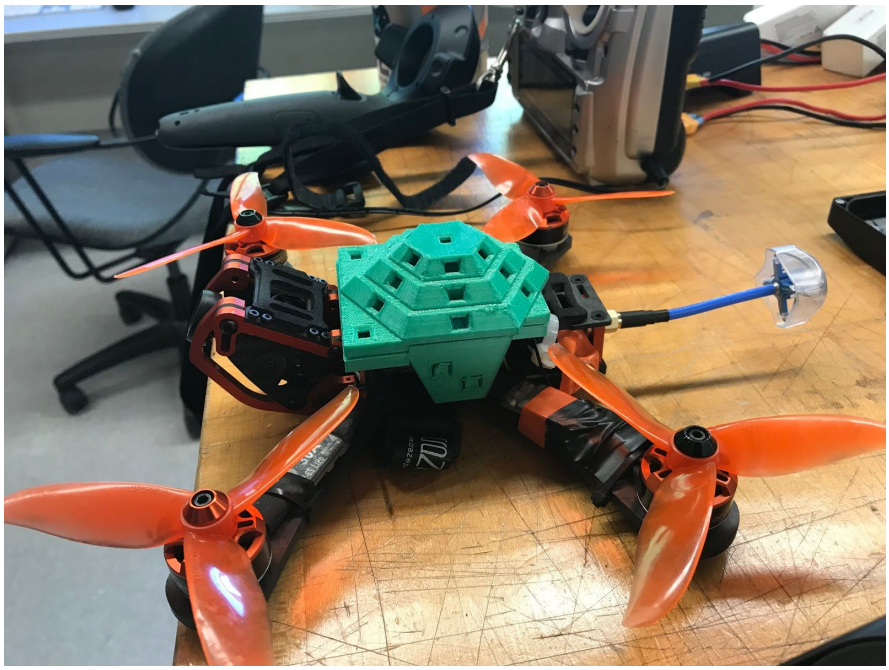

Figure 38 - Final Mounted Shell on Chameleon (left side)

4 - Hardware List and Funding Examination

Item	Description	Cost
HTC Vive	Complete system with two controllers, headset, hub, two light houses and wireless system.	$600
ValveVR HDK Development set	Hardware set as described in the HDK section above	$595
VR capable computer	Needed to run the system. Graphics card must meet the minimum specifications for ValveVR	~$1500
Chameleon Quadcopter	Custom racquad build	~$500 with varying support materials such as radio and batteries
	Total Cost	**$3,195**

This project was only made possible by the contributions of many generous people. The research team at LogMeIn contributed use of the HTC vive. The EE department provided free 3d printing. The capstone and working with Lynne Slivovsky provided the computer in which all simulations were run and the vive connected. The quadcopter and development boards were provided by family members and out of pocket expenses.

5 - Project Conclusion

Overall this system provides interesting possibilities for further development. Further integration could include live communication to the onboard flight controllers for system in loop testing. Additionally matlab models can be used to generate desired paths, the flight can be done, and the capture system could verify against this to provide insight. The most immediate applications that will be pursued is better Unity integration with other tools to reduce the amount of manual copying of data as well as UI/UX experience in the VR experience. The enclosure does need some more work to ensure that all components fit in correctly to ensure correct calibration.

The engineering requirements were met in that the system has been tested to be within a mm ^3 accuracy, is mobile, and is under $1000 for the Vive tracking system itself. One issue that needs to be fully explored is the use of the lighthouse boxes outdoors in a UV/IR rich environment in the sun.

Overall the system has proved to be a rich test bed for future development. The issue was never tracking itself, but what tracking could provide for future projects at which Valkyrie tracking has proven itself to work for research needs.

6 - Demo Photos

Figure 39 - Final Demo with all components and tested platforms

Figure 40 - Final Demo setup with the HTC Vive and VR computer

7 - Works Cited

{1} http://flyingmachinearena.org/

{2} Landry, B. (2015). Planning and control for quadrotor flight through cluttered environments. (Master's Degree Thesis, Massachusetts Institute of Technology).

{3} https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/

{4} http://charmedlabs.com/default/pixy-cmucam5/

{5} https://en.wikipedia.org/wiki/HTC_Vive

{6}https://www.roadtovr.com/analysis-of-valves-lighthouse-tracking-system-reveals-accuracy/
{7} https://trmm.net/Lighthouse

{8} https://www.triadsemi.com/product/steamvr-tracking-hdk/

{9 } https://partner.steamgames.com/vrlicensing