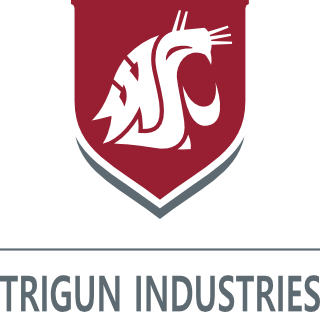
Precision Drone Landing in GPS-Denied Environments

 + 

Nikita Chrystephan

David Brownell

Christian Hunter

CptS 423 Software Design Project II

Fall 2020

December 20, 2020

**TABLE OF CONTENTS**

[**I.**](#_30j0zll) **Introduction** 4

[**II.**](#_1fob9te) **Team Members & Bios** 4

[**III.**](#_2et92p0) **Project Requirements Specification** 5

[III.1. Project Stakeholders 5](#_tyjcwt)

[III.2. Use Cases 5](#_3dy6vkm)

[III.3. Functional Requirements 5](#_1t3h5sf)

[III.4. Non-Functional Requirements 5](#_4d34og8)

[**IV. Software Design - From Solution Approach**](#_2s8eyo1) 6

[IV.1. Architecture Design 6](#_17dp8vu)

[IV.1.1. Overview 6](#_3rdcrjn)

[IV.1.2. Subsystem Decomposition 6](#_26in1rg)

[IV.2. Data design 6](#_lnxbz9)

[IV.3. User Interface Design 6](#_35nkun2)

[**V. Test Case Specifications and Results**](#_1ksv4uv) 7

[V.1. Testing Overview 7](#_44sinio)

[V.2. Environment Requirements 7](#_2jxsxqh)

[V.3. Test Results 7](#_z337ya)

[**VI.**](#_3j2qqm3) **Projects and Tools used** 8

[**VII.**](#_1y810tw) **Description of Final Prototype** 8

[**VIII.**](#_4i7ojhp) **Product Delivery Status** 10

[**IX.**](#_2xcytpi) **Conclusions and Future Work** 10

[IX.1. Impact Analysis and Ethical Issues 10](#_1ci93xb)

[IX.2. Limitations and Recommendations 10](#_3whwml4)

[IX.3. Future Work 11](#_2bn6wsx)

[**X.**](#_qsh70q) **Acknowledgements** 12

[**XI.**](#_3as4poj) **Glossary** 13

[**XII.**](#_1pxezwc) **References** 13

[**XIII.**](#_dwz0g8eitbzx) **Appendix A – Team Information** 14

[**XIV.**](#_147n2zr) **Appendix B - Project Management** 15

# Introduction

Client: Trigun Industries

Contact Information:

Collin Merriman - [merrimac@wsu.edu](mailto:merrimac@wsu.edu)

Phillip Whitworth - [phillip.whitworth@wsu.edu](mailto:phillip.whitworth@wsu.edu)

Mentor: José Delgado-Frias

Contact Information: [delgado-frias@wsu.edu](mailto:delgado-frias@wsu.edu)

Trigun Industries requires a self-contained drone landing system that uses affordable and off-the-shelf hardware. The internal project lead at Trigun Industries is Colin Merriman, and he has been the primary contact for all issues regarding implementation details and technical support along the way. He has also supplied a drone platform to use for the project.

The primary goal of this project is to develop an autonomous landing system for a quadcopter drone that relies primarily on a single camera. The landing system must have high accuracy and reliability and must be self-contained. Weight and power limitations will restrict this project to making use of low-power single-board computers.

Our approach ties together several existing frameworks and libraries to link together the drone’s flight controller with the camera. We use image recognition software to read a QR code on the landing target and use a machine-learning regression model to estimate the relative position of the drone to the target. Once we have obtained the relative position estimate, we feed it into a noise filtering algorithm. The filtered result is then passed into a PID controller which guides the drone toward obtaining zero relative position difference to the landing target.

# Team Members & Bios

## Nikita Chrystephan

Nikita Chrystephan is a computer science student at WSU. He was an ambitious programmer even before matriculating at WSU. He created several software tools for employers before starting school, such as a tool using google sheets and javascript to generate scannable QR codes automatically to streamline order picking and packing After that, he expanded the project to create an inventory tracking system to aid reporting and auditing of inventory.

After joining WSU, Nikita has completed several more notable projects. He created a recommendation system to match career questions to best qualified professional respondents using Python. He has also created a web scraper to scrape YouTube pages for metadata and comments with database, statistics reporting, and visualization using Python and SQL. The purpose of this project was to build a connections graph between recommended videos.

Autonomous vehicles are one of the ‘near future’ technologies that originally inspired Nikita to pursue a computer science degree, and he hopes to pursue a career working with related technologies after his educational career. Nikita is responsible for development of the simulation environment and noise filtering components of this project, and contributed to the control loop implementation.

## Christian Hunter

Christian Hunter is a fourth year computer science student. He has interests in mobile application development as well as embedded programming. His initial foray into the programming world was a class on robotics. After this he decided to go a little deeper into the embedded world and created a few projects using arduino. After beginning to attend WSU, Christian began to become more interested in the programming side and has since found an interest in mobile application development. The most notable project he has completed in app development is an app which allows users to track progress in a 6 week powerlifting program. It will calculate reps and sets for the exercises specified by the program. Other interests include weightlifting, wakeboarding, skiing, and video games. Christian is responsible for setting up the continuous integration platform, QR code generation script, and some minor additions to the control algorithm.

## David Brownell

David Brownell is a computer science student in senior year. His interests include 3D printing, electronics, and motorcycling. Over the past summer, David took an internship with Meter Group, focused on back-end web development. His skills include Python, Django, Linux, and Haskell. In this project, he is responsible for the displacement detection algorithm, as well as training the regressor used therein. He also performed the original redesign from a synchronous to an asynchronous main loop.

# Project Requirements Specification

Trigun Industries, the client and primary stakeholder, requires a reliable and self-contained autonomous drone landing system. The client also requires a proprietary implementation to achieve market advantage, so we have avoided including dependencies with restrictive free software licenses.

The client requirements are specific. We must design a drone landing system that can land on a predefined 1 by 1 meter target without the aid of GPS or other remote resources such as radio homing, infrared, etc. Additionally, depth sensing using sonar, radar, lidar, or multi-camera configurations are also prohibited. The method of operation has been restricted by client requirements to use only a single camera with support of internal IMU sensors for positioning. The client has supplied a custom quadcopter drone platform that uses a Pixhawk flight controller with ArduCopter firmware on-board. The supplied drone is about 24 inches in diameter.

The project requirements from our client state that our software should take over flight control once the drone has reached a point near the landing pad, roughly 10 meters above it. Our software should guide the drone from this point onto the center of the landing pad, with a margin of error of +-5cm from the center of the landing pad. The final requirement is that the software be robust and repeatable, with an ideal 99.9% landing success rate. A successful landing is one where the drone settles on the pad the margin of error and is facing the right way up.

## Project Stakeholders

Trigun Industries is the client and primary stakeholder. It is their need for a functional, reliable, self-contained, and autonomous drone landing system that we have strived to meet first.

Future customers and beneficiaries are also potential stakeholders in this project. These include package delivery services, private security firms, mapping and surveying organizations, and national security customers, all of whom require highly reliable solutions.

Proliferation of autonomous drones has the potential to raise concerns with the public, but the public (as a group) will also be beneficiaries of some autonomous drone applications, such as package delivery. We will consider the at-large public to be tertiary stakeholders for the purposes of this project.

## Use Cases

Imagine a GPS deprived environment (such as a dense cityscape) where there is still a need to fly drones. For instance, we could imagine a user having a number of drones working on surveillance. These drones will eventually need to land at a home base to recharge, and do so accurately. This is a task that can be difficult in an environment where GPS is available, let alone one without GPS. Our solution aims to solve just this problem. We have accurately and precisely solved this issue by creating a landing pad which allows a bottom-facing camera to track the exact position where we need to land, and then adjusts our drone’s perceived relative position accordingly. Then, we command the drone to move in the direction of the landing pad. Our solution allows us to land in nearly the same position every time.

## Functional Requirements

* 1. **Autonomous Drone Landing System**

In an effort to reduce costs and computational load, Trigun Industries requires the drone landing system (DLS) to be driven by visual input from a single camera. Potential future customers include commercial drone delivery services, private security contractors, agricultural surveyors, and national security clients. All of these clients require a foolproof, accurate, and reliable implementation to minimize the need for human intervention and especially to minimize liability and maintenance expense risks. National security clients in particular require a drone that is resistant to interference. For this reason, a self-contained solution that does not require communication with off-board resources is required.

* 1. **Self-contained system**

Our DLS must operate without remote support because it cannot depend upon reliable connectivity in all potential use cases, such as remote landing zones. This requirement is necessitated by the client demands and potential national security customer demands. Since the hardware and software must all be on the drone, we have significant power and weight limitations for our hardware. We have chosen to use a Raspberry Pi single-board computer along with a lightweight camera as our setup.

* 1. **Design Parameters**

The design parameters given by the client are that we must acquire the landing zone from 10 meters elevation above ground and put the drone on the ground on target with a margin of ±0.05 meters. In the event that the QR code cannot be seen by the drone, a search sequence is initiated to find the code nearby.

* 1. **Transfer of Control**

The drone landing system must allow itself to be activated by other systems on the drone and must take over control in midair. Therefore, interfacing with the other drone flight systems is critical. This requirement comes from the client parameters that state that the drone will navigate to the approximate area of the landing zone using other systems.

## Non-Functional Requirements

1. **Simulated Testing**

Trigun Industries requires a highly reliable Drone Landing System due to its clients’ use cases. To validate that the landing system really is reliable, we have established a method of testing which allows us to test the drone and our software entirely within simulation.

1. **System Reliability**

The system must be capable of landing within ± 5cm from the target with a 99.9% success rate. This requirement is explicitly laid out by the client because of anticipated commercial and governmental customer needs. As of now, we have not yet achieved such a high success rate. We estimate that our current success rate is around 99%.

1. **Availability**

The DLS must be capable of running in all weather conditions wherever the drone can fly, including at night. This requirement must be met by future work. We have not validated our solution against all possible weather conditions, due to the limitations of our simulation software.

1. **Landing Pad**

The landing pad used will be a 1x1 meter pad that will have an image to allow the drone to properly orient itself and land on the correct spot. The pad must not rely on active components such as lighting, wireless radios, ultrasonic beacons, or other powered landing aids, and the LZ is assumed to be unpowered. This requirement comes from the client, anticipating governmental requirements. We have selected a QR-code-based landing solution for this application.

1. **Visual Recognition:**

The system shall be capable of tracking the landing pad and landing in the predetermined orientation on the pad.

1. **Delivery:**

We have agreed with our client to package and deliver the software in the form of a git repository including installation instructions by 01/03/2020.

# Software Design - From Solution Approach

## Architecture Design

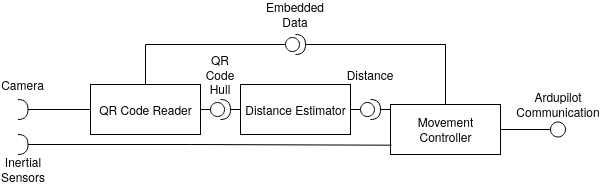
### Overview

As this is an embedded project, it’s difficult to assign a typical architectural design pattern. We found that an asynchronous event loop served our needs best. Our overall design might be classified as a Pipeline with some intermediate state. The individual components are:

The QR Code Reader is responsible for two distinct, though related tasks. Namely, it must detect the location and size of the QR code in an image retrieved from the camera, and also extract the data stored in the code.

The Distance Estimator uses the location and size of the QR code, as determined by the QR Code Reader, to make a guess at where the drone is located relative to the target. We have applied a neural network to this task, as we have exhausted conventional methods to solve the problem from a strictly mathematical perspective.

The Movement Controller decides how to fly the drone. It uses information produced by other subsystems and issues commands to the Ardupilot microcontroller which is controlling the low-level flight systems of the drone. We have implemented a smoothing algorithm to help make the output from the Distance Estimator more consistent.



*UML component diagram for our architecture design; notice that the data flows in one direction only.*

### Subsystem Decomposition

### QR Code Reader

#### *Description*

The QR Code Reader is responsible for finding the QR Code(s) in an image retrieved from the camera. It must locate QR codes as well as extract the data stored in them.

#### *Concepts and Algorithms Generated*

QR code reading is in large part a solved problem. There are several libraries to choose from, but we chose ZBar because of its reliable performance in our testing. One drawback of PyZBar is that by default it returns the corner points of the qr code (which we use to estimate its position) in an arbitrary order. Thankfully, the underlying ZBar library does not have this problem. There are customized versions available that provide a reliable ordering.

#### *Interface Description*

##### Services Provided

Service name: QR Code Hull

Service provided to: Distance Estimator

Description: The hull is the outline of the QR code. Its size and shape is used to determine the approximate distance from the target. It is derived from an image retrieved from the Camera.

Service name: Embedded Data

Service provided to: Movement Controller

Description: The hull is the outline of the QR code. Its size and shape is used to determine the approximate distance from the target. It is derived from an image retrieved from the Camera.

##### Services Required

Camera

### Distance Estimator

#### *Description*

The distance estimator is responsible for calculating the estimated distance from the QR code/landing platform to the quadcopter. In practice, it creates a three dimensional distance vector.

#### *Concepts and Algorithms Generated*

*The Distance Estimator is able to use the QR code hull from the QR Code Reader to determine an estimate of how far away the drone is from the target, as well as the angle between the target and the drone.*

#### *Interface Description*

##### Services Provided:

Service name: Distance

Service provided to: Movement Controller

Description: The Distance Estimator provides its guess to the Movement Controller so that the latter may make a more informed decision as to where to land. This is important, as, without the distance estimation, the drone is unable to tell how close it is to the target, or what angle the target is at.

##### Services Required:

QR Code Hull

### Movement Controller

#### *Description*

The movement controller is responsible for determining which direction and how fast the quadcopter should move based on the estimated distance and position relative to the landing platform.

#### *Concepts and Algorithms Generated*

Because a major goal for our software is to land the drone in a GPS-denied environment, we will not be able to take advantage of Ardupilot’s built-in landing and control algorithmsthat rely upon GPS*.* This means we need to develop our own control algorithm to determine the speed of the motors. We have chosen to use a set of PID controllers, which are a well known and efficient control algorithm. This algorithm takes the estimated distance vector as input and based on a proportional, derivative, and integral calculation, generates new movement targets for the drone.

##### Services Provided:

Service name: Ardupilot Communication

Service provided to: Ardupilot

Description: The movement controller is a service which changes the drone’s movement based on the estimated distance from the landing platform. The input is the estimated distance from the landing platform and also the data from the onboard sensors found in the pixhawk flight controller. As output the controller provides three-axis movement targets in order to control the drone.

## Data design

As our project makes very sparse use of persistent data, it is largely unnecessary to design such data long in advance. However, the one place where we may need to think about data design in advance is the QR code landing pad.

Testing has determined that we need three different levels of landing pad. Furthermore, we are able to store about 6 bytes of readable data per code. At minimum, we would need two bits to store the code level, since we can pre-program the drone with the level sizes. This leaves 46 bits free for landing pad identification. Future work could focus on making use of these bits to verify that a landing pad is authorized.

## User Interface Design

Since the drone is designed to be used autonomously, there is, by nature, limited user interaction with the drone. However, some interaction is necessary. We identified that someone must install the software, and a user must be able to pair the drone with a landing pad. We have chosen to place text files in the drone software folders that provide the codes the drone is searching for. A secured solution may be a focus of future work.

Much of the project has focused on refining the algorithm performance within a virtual environment. Therefore, it is important that the software stack for this environment also be documented so that it can be recreated in the future. We have selected an Ubuntu-based distribution as the base software platform, and we have built the test environment upon that. Details will be included with the final project hand-off packet. We have also included an installation script for software dependencies within the development environment.

# Test Case Specifications and Results

## Testing Overview

Our drone landing system is unlike many other projects in that it is an integrated system which includes hardware and software components and relies upon externally generated data to function (camera view). As a result, we must take a unique approach to testing.

Some portions of our codebase work by receiving real-time inputs and providing outputs that depend upon not only the most recent input, but also the recent history of inputs. For these functions, unit testing alone is not sufficient. Instead, we have opted to develop a virtual testing environment using Unreal Engine 4, Airsim, and Ardupilot. This virtual environment provides simulated inputs and outputs that allow us to accurately measure the performance and precision of our control system. We run real-time simulations that provide instant feedback to code changes. In addition, we can compare the measured position of our drone against the actual position and we can measure the resulting landing accuracy and precision.

This approach replaces automated testing for all but unit tests, because it is not possible to run this simulation on a low-powered computer or a server without a GPU. For this reason, we are opting not to pursue automated integration testing.

We were not provided a drone or a space within which to land during the refinement period of the project, so we have not begun real-world tests. However, our simulated environment is very sophisticated and uses many of the same software packages found on the drone itself.

Since much of our project depends upon maintaining sufficient performance on embedded hardware, we have maintained a branch of the project that runs on the raspberry pi test environment. While we cannot use this to run our simulated environment, we can use it to ensure reliable performance and to ensure software compatibility.

## Environment Requirements

Our hardware and software requirements have been finalized. The hardware and software stack includes the following:

* Custom quadcopter drone supplied by client - roughly 24 inches diameter.
  + PixHawk 2 flight control module
    - Running ArduCopter firmware
* Raspberry Pi 4B
  + Running Raspberry OS
* Python 3
* OpenCV
* DroneKit
* SimplePID
* PyZBar

Requirements for the simulated environment are different. We run the simulation on a desktop computer. Therefore, the requirements are as follows:

* Personal computer
  + Discrete GPU (AMD or Nvidia)
  + Ubuntu-based Linux OS
* Unreal Engine 4
* Python3
* OpenCV
* Airsim
* DroneKit
* SimplePID
* PyZBar

## Test Results

## Unit Testing

To perform unit testing, we are using the Python language’s “pytest” framework in conjunction with Gitlab’s gitlab-runner software running on an independent server. At present, we only have a handful of unit tests, because of the difficulties explained above. Units are difficult to isolate in this system.

## Integration Testing

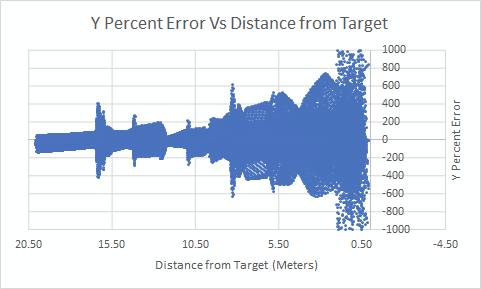
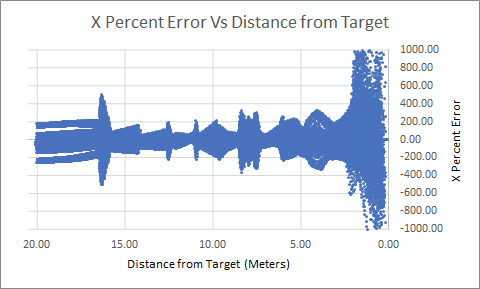
Our approach to integration testing in this project is to log all of the data we can from inside the application while the simulation is running. In this way, we can collect data that represents real-world operation and then perform analysis on the data we’ve collected afterward. This form of integration testing has to date been run without including any drone hardware in the loop, except for our Raspberry Pi. Connecting the Raspberry Pi to a PixHawk is the next step and one of the final steps before making delivery to the client.

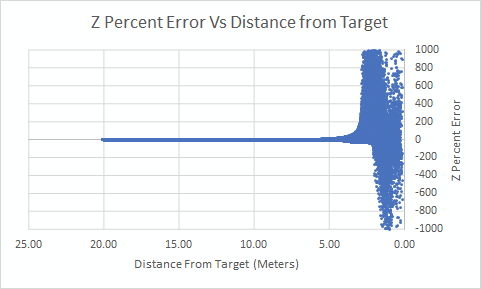
System Testing

## Functional testing:

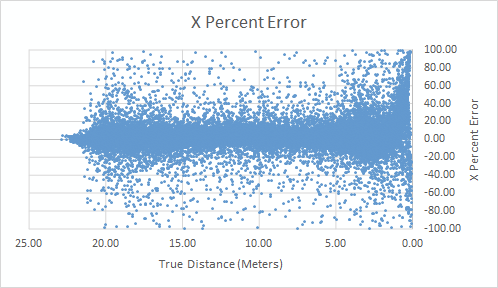
Our software has achieved several functional requirements so far. Our software can currently retrieve images from the drone camera and command the drone’s flight microcontroller to move in response. In short, we have achieved closed-loop control. We have tuned the PID control loops, the guidance prediction model, and the noise filtering algorithm to improve landing performance and accuracy. We have reached the point where we can reliably land within the client specified margin of error. We have compiled some visuals that show how the performance has improved over time.

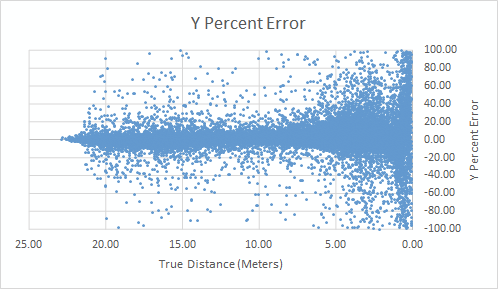
Our distance estimation system uses a trained regression model. Previous versions had comparative deficiencies in training and neural network design and suffered from high noise floors and reduced performance as the distance to the target decreased. That loss in performance is evident from the following graphs depicting simulated tests against the first version of the regression model.

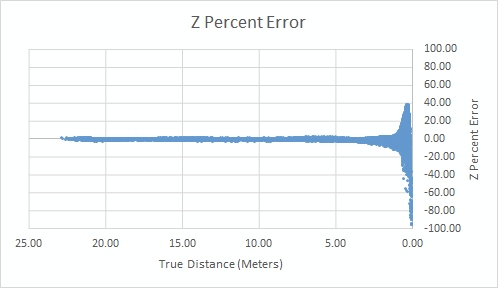




The synthetic test results from the latest (V3) version of the model show drastic improvements. Whereas the previous model had moderate accuracy at medium ranges, the prediction was nearly useless at short ranges. The new model has better accuracy throughout the range, as shown in the graphs following. This was achieved by randomizing the inputs and therefore randomizing the order. In particular, greater care was given to the low end of the distance scale. Additionally, the neural network structure was redesigned to add more and smaller internal layers. Notice that, while there are still outliers, the percent error varies less throughout the entire data range and the majority of the percent error is now within 100 percent, where before it frequently exceeded 1000 percent.



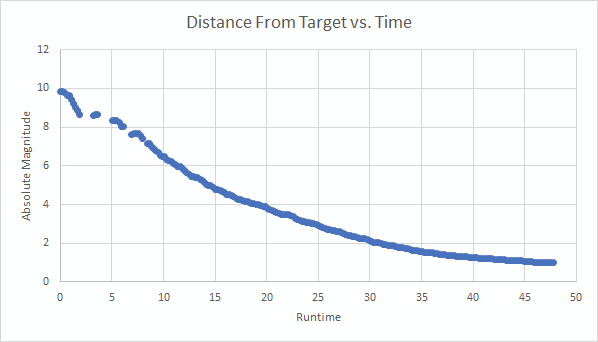
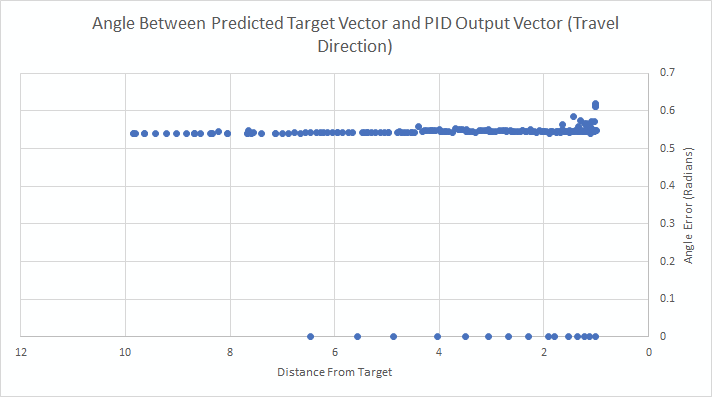


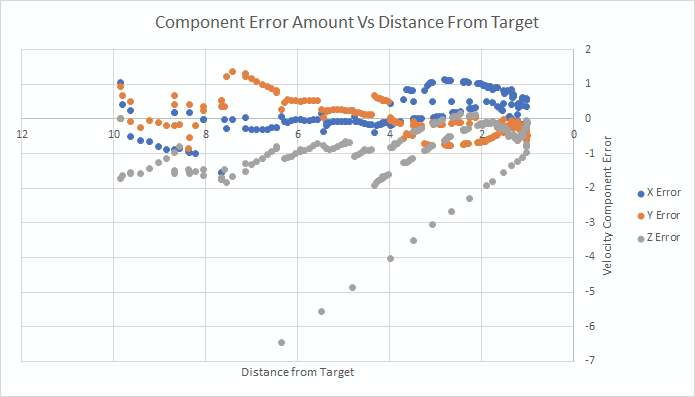


## Performance testing:

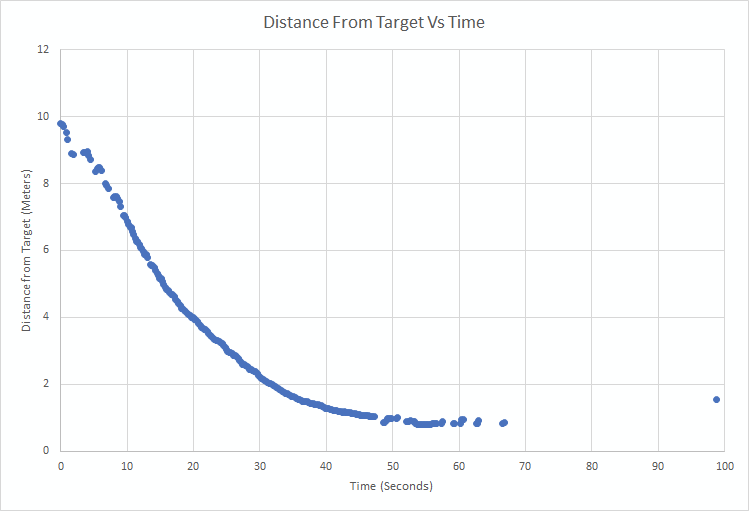
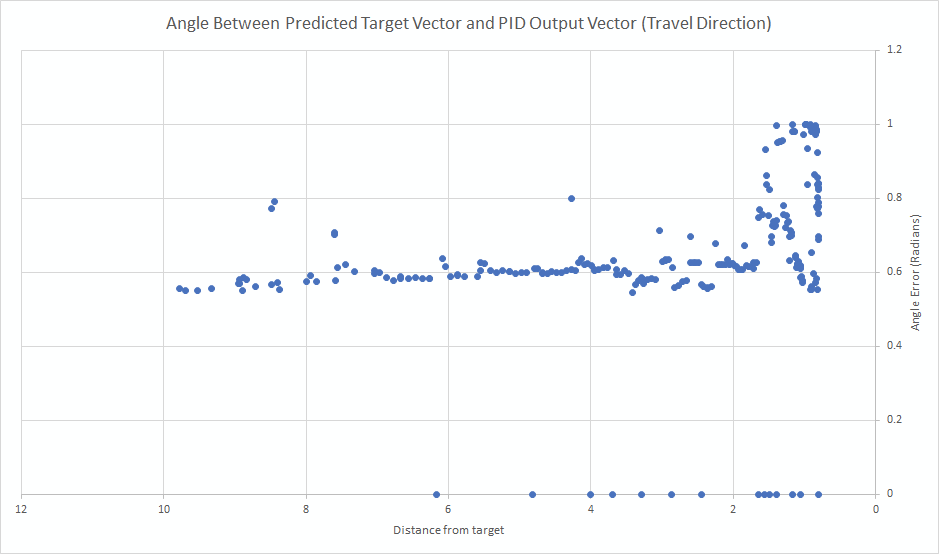
Software simulation is used to test the accuracy and reliability of the drone landing system. The live simulation measures the time taken for landing and the resulting distance from the landing target. We also log the position of the drone at every time interval during the drone’s run as well as its velocity vector and predicted position. Results from successive runs are used to build a theoretical performance profile.

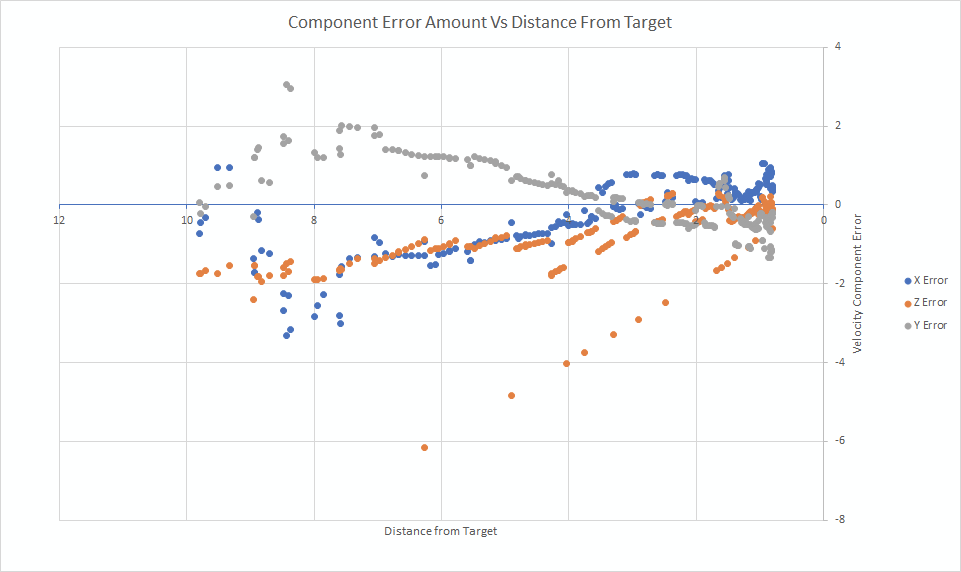
We have collected data from numerous runs of our drone landing software in simulation, and we have included the results from a handful of runs below. First, we start with runs that use our V1 distance estimation model with our V1 noise filter. The three graphs below show data collected from a successful landing within the margin of error for client requirements. Unfortunately, this early version of our algorithm was not reliable enough and this result occurred only about 10 percent of the time.



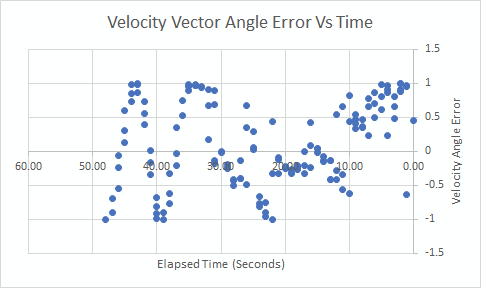
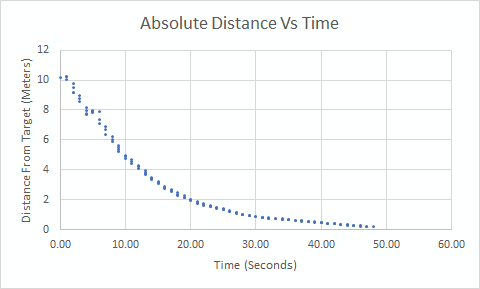


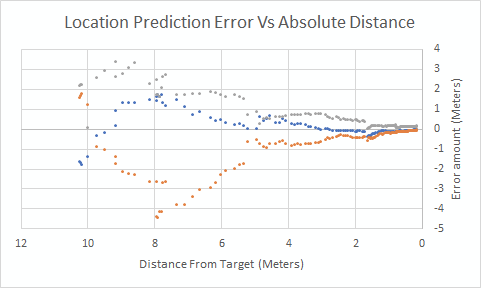
The second set of graphs (below) is from a landing that was on the landing pad (within 0.5 meters of the target) but was not within the client specifications margin of error (0.05 meters). This is a typical result from the V1 software. The final distance from the target can be seen in the second graph as the blue dot at the far right.





Next, we present data from the latest version of our software. This uses an enhanced distance estimation model with less distance bias, lower error, and an improved filtering algorithm that calculates a time-weighted average of recent target estimates. PID performance has also been refined.





The new results show that the angle between the predicted target location and the direction the drone is commanded to move is consistently high. This is to be expected, since we have moved away from using a PID control for the vertical axis, and we have tuned the horizontal pid controls to be more aggressive in centering the drone. Prediction accuracy converges much faster than before and converges to near zero error.The drone also follows a smooth path toward the target and lands within the client specified error margin and lands in less time than before.

Algorithm efficiency and performance are tested using a Raspberry Pi and a PixHawk CubeBlack to perform HITL simulations. The virtual environment measures accuracy and reliability of the algorithm, but the goal of HITL testing is to measure speed and consistency. We measure the time between command updates and the variation between these times. These low-power devices are an order of magnitude less powerful than the desktop computers we are developing on.

Our client is in the process of arranging testing plans and preparing a target.

## User Acceptance Testing:

The client has been shown a demonstration of the project running in a simulated environment and was generally positive about the current state of the project, but noted that we still need to test on hardware before considering the project to be at completion. The client has also requested that we ensure that all developed software be well documented with instructions on how to install and use the tools we have developed. Much of the documentation has already been written, but we will continue to work on it until the project is delivered to the client in early January.

# Projects and Tools used

|  |  |
| --- | --- |
| Tool/library/framework | Purpose |
| PyZBar | Barcode reading |
| pymavlink | Drone microcontroller commanding |
| scikit-learn | Machine learning model generation |
| NumPy | general vector library |
| OpenCV | Image processing |
| Shapely | Computational geometry |
| dronekit | Drone inputs |
| simple\_pid | PID controller |
| PILlow | Image processing |

|  |
| --- |
| Languages Used in Project |
| Python |
| Shell script |
| YAML |
| JSON |
| Markdown |

# Description of Final Prototype

## Usability

Our prototype is different from many projects because it is intended to be integrated into an existing hardware platform. Not just any hardware platform, but an autonomous drone. The primary intention of the project is that it *doesn’t* require human interaction. Therefore, we don’t have a specific user in mind except the drone itself. Consequently, there is no user interface.

Instead, we interface with the software that the drone is running already. We can receive updates on the drone state at any time and we can also send updates to change the drone state. We use this bi-directional link (MavLink) to command the drone to land based on the external feedback our module receives (Raspberry Pi with camera module) and also to detect when we should begin the landing process.

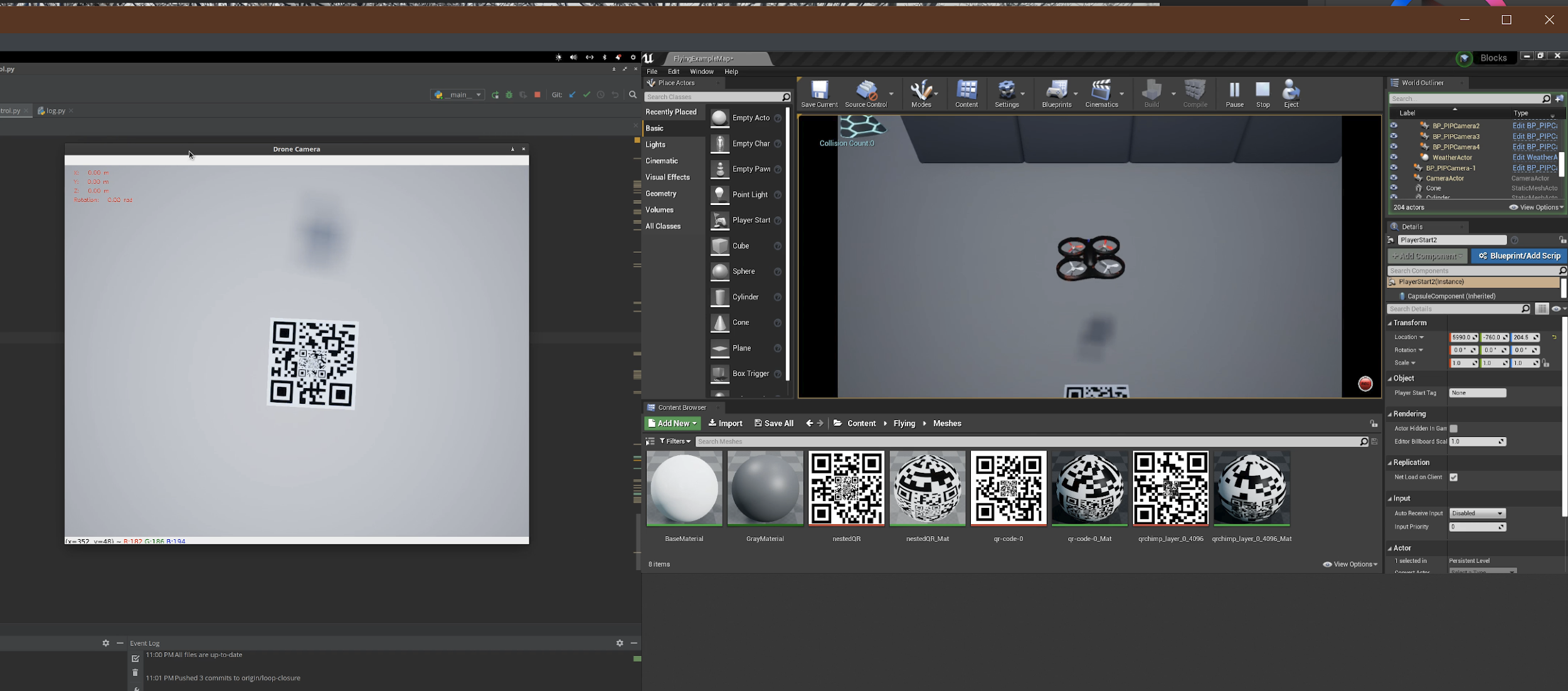
There are only two times the user should have to interact with our software. The first is when they install it on the target drone. They will need to modify the connection parameters of our code so that it connects to the drone firmware instead of the simulated environment (which is typically on the same computer). The second time they will need to interact with it, is when they want to enable authentication of landing pads. This will require adding a file to the project code installation folder that contains the codes the drone is looking for. Note that landing zone authentication is not currently implemented, so adding pre-authorized landing codes is not currently required.

## Use-Case

The autonomous landing system we have developed is designed to be triggered by the drone when the drone decides it has reached the appropriate position. Our landing module will automatically take control of the drone when launched and guide the drone toward the target as previously discussed in this document. The drone developer can opt to either have our software run concurrently while the drone flies and listen for a specific flight condition to activate or they can opt to launch our software when they determine it is time to land.

## Testing

Below is a screenshot from within our virtual testing environment. The data that we collect from this environment has been discussed extensively already. This image shows what a typical landing sequence looks like within the environment, where we collect data on position prediction accuracy, landing accuracy, time taken, and more. In the following screenshot, we are ensuring that the drone lands on target, which is a project requirement. We can also observe the drone in flight to see if it exhibits any peculiar flying characteristics. The virtual drone may not have the exact same flight characteristics of the drone that this software is eventually deployed on, but our implementation is largely drone agnostic since it commands the drone in a high-level way.



# Product Delivery Status

We have not yet delivered the project to the client. We plan to do so on January 3rd.

At delivery we will present our git repository to the client, as well as our documentation and borrowed hardware. Installation instructions will be provided in the project’s README file.

# Conclusions and Future Work

## Impact Analysis and Ethical Issues

Drone technology is an area of intense controversy, in no small part due to America’s overseas drone bombing campaigns. These drones are not at all the same kind of technology our project contributes to, however. First, they are fixed wing aircraft. Second, they are (mostly) not autonomous. The performance characteristics of rotary drones such as the quadcopter this project is based upon do not lend themselves to direct combat missions due to their minimal range and relative lack of speed or maneuverability.

The above does not mean that there are no ethical dilemmas associated with copter drones, however. Copter drones are ideally suited to surveillance or package delivery tasks within a small range. Whenever the idea of surveillance arises, ethical concerns also arise. Will the technology be used to invade people’s privacy or otherwise invade their rights, or will it be used to rightly defend private property or secure locations? Package delivery drone technology could also be mis-used to deliver explosives or other unwelcome cargo. However, much like the surveillance use-case, the positive or negative impact of the technology has more to do with the goals of the people who use it than it does with the technology itself.

It is insufficient to simply declare that because a technology can have both positive or negative implications that the creator of the technology bears no responsibility for its ultimate application, however. Certainly the founders of Napster were not held blameless for how people used their file sharing service.

The software we have developed here, which is essentially a drone guidance system, is dependent upon a landing pad that is pre-defined and well known before the drone ever launches. It is also assumed that the landing pad is static. It would be difficult to adapt this software to provide guidance capabilities for a drone or other airborne vehicle that aims to seek out a different kind of target, mostly due to how specialized our regression model is and how much the rest of our implementation depends on the assumptions we have made about our landing target. With all of this in mind, the ‘homing’ capabilities we have developed here have limited use outside of their intended use case.

The intended use case is landing a drone safely, which is not necessarily a goal of a nefarious actor. As described above, the software is minimally adaptable for tracking other kinds of targets, so its use in enabling surveillance is likewise limited.

This landing technology could be adapted for use in a copter drone that pursues other undesirable goals, but we feel that the risk of this technology directly enabling harm is low and therefore the ethical implications are minimal.

## Limitations and Recommendations

The prototype that we have developed has been extensively tested and refined in a simulated environment. It performs the client goals of landing a drone accurately and reliably within that environment. However, the body of tests have not yet accumulated to a level where we can guarantee that it meets the 99.9 percent reliability benchmark. We can say that it is roughly at the 99 percent mark, however. The virtual testing is unfortunately useless if the drone does not behave similarly in the real world. Because we have abstracted away the direct control of the drone and left that in the hands of the flight controller, we suspect that any well-tuned ArduCopter based drone should be able to implement this software with minimal changes. We cannot, however, guarantee it. Therefore, more testing in the real world is recommended.

Testing in adverse weather conditions was originally a goal of this project, though it was not expressly stated in the original requirements brief. Our test environment is able to simulate rain, which does not have a major impact on (virtual) drone landing performance. We suspect that water and (non-virtual) drones do not mix well, however. Unfortunately, our virtual environment does not support wind. This is an important real-world concern, since wind often affects drones. Additional testing on the effect of wind on the landing algorithm is needed.

A stretch goal of the original project requirements was to authenticate the landing pad so that the drone would be sure to avoid unauthorized or improper landing locations. While the landing pad specification we created does embed data that can be used for landing pad authentication, the current prototype does not make use of it. This is listed in the future work section below.

## Future Work

Based on the work which our team has accomplished there are many directions for expansion. The immediate next steps of the project would be real world testing and tuning adjustments. This is to ensure the drone can maintain it’s landing accuracy at a similar consistency to the simulation. The drone should also be tested in varying weather conditions to ensure the landing algorithm is able to compensate well enough in a less than perfect environment. With the drone landing, development can progress to determining a method of verifying the landing pad. In its current configuration the landing pad contains three nested QR codes which can each contain a maximum of 6 bytes of raw data. Whether the QR code itself is used or some external form of validation from the landing pad is used, a verification scheme is needed to ensure each drone lands on the correct pad. In parallel with the landing verification it is necessary to determine a method of externally activating our software upon reaching the landing zone. Currently, for testing purposes our software arms and takes control of the drone before takeoff. In the real world use case the flight controller needs to be able to initiate landing upon the completion of its predetermined flight route.

# Acknowledgements

Thanks to:

* Colin Merriman, for providing us with this opportunity and guiding us along the way
* José Delgado-Frias, for being our mentor during this challenging time
* Aaron Crandall, for providing us with advice and guidance early in the project

# Glossary

|  |  |
| --- | --- |
| **ArduCopter** | Firmware that runs on a PixHawk and similar flight controllers for drones, airplanes, and other small vehicles and provides a framework for communication and control. |
| **CubeBlack** | Model name for the flight controller used by the hardware test platform (drone). |
| **HITL** | Hardware-In-The-Loop. HITL indicates that both our application software as well as the drone hardware is involved in running a simulated drone. This is a stronger requirement than SITL, but weaker than real-world testing. |
| **SITL** | Software-In-The-Loop. SITL indicates that our application software is used in running a simulation. This is a weaker requirement than HITL. |
| **Raspberry Pi** | A single-board computer with a compact size and power draw between 3-4 watts, capable of running Linux based operating systems and arbitrary code. |
| **PixHawk** | Brand name for the manufacturer that manufactures the CubeBlack flight controller used by the hardware test platform (drone). |
| **PID Controller** | (Proportional, Integral, Derivative) A simple and efficient method of manipulating and maintaining a particular value. PID controllers are frequently used in drones to control movement and orientation. Other applications include cruise control in automobiles and temperature in climate control systems. |
| **QR Code** | A particular type of two-dimensional label readable by a machine. Other similar specifications exist, such as Data Matrix and Aztec Code. |
| **GPS** | a navigational system using satellite signals to fix the location of a radio receiver on or above the earth's surface. |

# XII. References

|  |  |
| --- | --- |
| **1** | R. Barták, A. Hraško and D. Obdržálek, “On autonomous landing of AR. Drone: Hands-on experience,” *The Twenty-Seventh International Flairs Conference*, 2014. |
| **2** | R. Barták, A. Hraško and D. Obdržálek, "A controller for autonomous landing of AR.Drone," *The 26th Chinese Control and Decision Conference (2014 CCDC)*, Changsha, 2014, pp. 329-334.  doi: 10.1109/CCDC.2014.6852167 |
| **3** | L. Goeller, “Drone Precision Landing using Computer Vision,” *Drone Precision Landing using Computer Vision*, 24-Jun-2018. [Online]. Available: https://pub.tik.ee.ethz.ch/students/2018-FS/SA-2018-21.pdf. [Accessed: 27-Jan-2020]. |
| **4** | S. Lange, N. Sunderhauf and P. Protzel, "A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments," *2009 International Conference on Advanced Robotics*, Munich, 2009, pp. 1-6. |

# 

# Appendix A – Team Information



Team members (above) from left to right: David Brownell, Nikita Chrystephan, and Christian Hunter.

# 

# Appendix B - Project Management

Throughout this project we had bi-weekly Zoom meetings with our instructor, José Delgado-Frias. These meetings helped to ensure that we stayed on track, as well as enabled us to resolve any blockers which were holding us up.

Our meetings with the client were considerably less consistent, our client is extremely busy so we would try our best to accommodate meetings when he was available, during these meetings we would usually demo our progress as well as clarify any requirements which we were unsure about. Our mentor also provided technical guidance on questions relating to practical flying of drones and provided examples of technologies that are already used in the space.

We also had weekly team meetings, during these meetings we made sure all team members were up to date on current project status. Also, discussed any issues that we were facing and determined what direction to take the project in the following week. These meetings were split between Zoom calls as well as text chat in Discord.

We used a feature of GitLab and discord to allow our Discord chat to receive notifications whenever a team member pushed new code to our Git repository, which helped us to keep track of what progress was being made and who was putting in work. Detailed Git commit messages helped us to communicate exactly what was worked on (changed) in each new commit.

Our team found that the weekly team meetings were the most important because they helped us to set short term goals and accomplish more work on a week to week basis.

Our go to tools for team interaction during this project were Zoom and Discord. Zoom allowed us to easily set up video conference meetings with our instructor, client, and each other. We also relied heavily on Discord for quick text chats. This allowed us to send quick chats to each other without setting up a meeting.