SIMULATION DEVELOPMENT, PATH PLANNING, AND LINEAR QUADRATIC GAUSSIAN PRECISION LANDING FOR UNMANNED TRAFFIC MANAGEMENT OPERATIONS

A Thesis
IN
Mechanical Engineering

Presented to the Faculty of the University
of Missouri–Kansas City in partial fulfillment of
the requirements for the degree

MASTER OF SCIENCE

by
JUSTIN NGUYEN

B.S., University of Missouri-Kansas City, MO 2020

Kansas City, Missouri
2022

SIMULATION DEVELOPMENT, PATH PLANNING, AND LINEAR QUADRATIC GAUSSIAN PRECISION LANDING FOR UNMANNED TRAFFIC MANAGEMENT OPERATIONS

Justin Nguyen, Candidate for the Master of Science Degree

University of Missouri–Kansas City, 2022

ABSTRACT

Unmanned Air Systems (UAS) are heavily utilized in various missions for both military and consumer applications. Because of their immense popularity, the Federal Aviation Administration (FAA) has derived a notional architecture for an Unmanned Traffic Management (UTM) system to regulate the airspace for UAS operations. In addition, NASA has conducted research on the Extensible Traffic Management, where the airspace will be shared by both manned and unmanned systems [16].

This thesis addresses the issue of the need for a High Fidelity Simulation Environment to test the interaction between simulated UAS with UTM and its Universal Service Suppliers (USS). The framework proposed provides extensibility for additions of different USS for testing protocols.

This thesis also contributes a hierarchical multi-UAS path-finding algorithm that

can be adjusted for different topologies as well as mimics standard route planning methods conducted in manned aircraft operations. To test this simulation 30,000 total Monte Carlo simulations were conducted for 10 to 100 UAS operators, with different prioritization techniques. In addition, the method was tested in the High Fidelity simulation.

Last this thesis provides a control and guidance law for tracking and precision landing of quadcopters for future applications of USS and industry applications, utilizing AprilTags with a Linear Quadratic Gaussian (LQG) law. Tests were conducted with the High Fidelity framework developed. Results indicate that the guidance and control law is robust from gust disturbances injected in the simulation realm.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Computing and Engineering, have examined a titled "Unmanned Traffic Management the Trilogy: Simulation Development for UTM Operations, Multi-Agent Path Finding Algorithm for UTM Operations, Linear Quadratic Gaussian (LQG) Design for ApriTag Tracking with a Quad-Copter" presented by Justin Nguyen, candidate for the Master of Science degree, and hereby certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Mujahid Abdulrahim, Ph.D., Committee Chair
Department of Civil & Mechanical Engineering

Travis Fields, Ph.D.
Department of Civil & Mechanical Engineering

Antonis Stylianou, Ph.D.
Department of Civil & Mechanical Engineering

CONTENTS

ILLUSTRATIONS

TABLES

ACKNOWLEDGEMENTS

I would like to thank Kevin Kowalik and Jared Anderson from HCI Energy and Daniel Campbell from Drop Drone for helping fund my research towards these topics. It was an absolute pleasure working with you guys and gaining perspective on the industry.

I would like to thank my advisor and mentor Dr. Mujahid Abdulrahim for being the positive character he always is. His continuing support and encouragement to better myself as an engineer is something I have always appreciated. I have not met a man as interesting ,a helluva pilot, and a controls engineer as him.

I would also like to thank Dr. Travis Duane Fields for introducing me to the world of robotics and programming. He was the first professor to make me realize there is more than HVAC and Manufacturing for a mechanical engineer as well as egging me to think outside of the box in solving problems. There is always more than one solution to a problem.

I would also like to thank Dr.Antonis Stylianou and Dr. Dianxiang Xu, for teaching me fundamental concepts of state space formulation, rotation matrices, and matrix math (Dr. Stylianou's CAE class) and the concepts of good software design principles and practices (Dr. Xu's Software class). You both challenged me in your courses but I gained a lot at the end of it.

I would like to thank my family particularly my mom, sister, brother, and recently passed grandma for reminding me to stay true to myself and reminding me to be a better man everyday. Hope you're looking down proudly at me grandma.

I would like to thank my FAST Lab colleagues for the positive and energetic work environment. Shoutout to Peter, Alan, Austin, and Francis for their hard work and commitment to get the job done, you guys are going places. I would also like to thank the PAVS Lab particularly Michael, Alec, and Simeon for their hardwork and determination. You guys always inspire me to keep on going when things are rough.

CHAPTER 1

INTRODUCTION

1.1 Background

UAS are utilized in numerous different applications such as military deployments, agriculture, disaster relief, and transportation of medical supplies [7]. Currently in the United States, the Federal Aviation Administration (FAA) has projected that by 2023, between 2 to 3 millions UAS will be deployed for various operations [1]. From these projections, there is a need of a proper air traffic management control system to provide safe operations. The FAA has written a Concept of Operations (ConOps) for this Unmanned Traffic Management (UTM) System providing the general framework of what services the UTM system should be able to provide, encouraging third parties support the operation of air services through connecting and sharing information with one another and with the FAA. In addition to UTM, NASA has conducted research of Extensible Traffic Management (xTM). In xTM, the airspace no longer be segregated between manned and unmanned but will be shared together [16].

1.2 Motivation

Although this notional architecture for UTM has been developed, there has only been a few live demonstrations of the application of the UTM, with testing still ongoing. This leads to the argument of the need of a simulation environment to provide numerous

and repeatable tests to evaluate the UTM's ability to *how* regulate and control numerous UAS for guidance and control of these systems. This simulation environment is to ensure that the protocols of the UTM and any third party service configured is robust enough to handle all situations, in a reliant manner. By building a robust simulation, it provides a safe environment to identify any potential failures prior to full launch of the service, mitigating risks and hazardous events from happening.

In addition the path planning for these UAS is an interesting task, since it is impractical to believe that these UAS all operate on the same firmware, software, and algorithms for control. This leads to the question of how path planning should be conducted for these UAS that abstracts the information of the UAS and plans for these paths accordingly.

Last landing is always crucial, and for UAS there is ongoing research for the development of charging stations for these systems. With these charging stations, it would allow UAS to charge during intermission of mission deployments. For these applications it would be crucial the precision landing protocol and its guidance be robust.

1.3 Summary of Objectives

The main objectives of this work are as follows:

1. Provide a High Fidelity simulation environment for Multi-UAS applications for testing of UTM protocols.

2. A multi-agent path algorithm that can be easily integrated into UTM applications for other UAS.

3. Implement a guidance and tracking law for UAS to conduct autonomous precision landing for UTM operations.

1.4 Summary of Contributions

The main contributions of this work are as follows:

- High Fidelity Simulation for UTM
    1. Designed software architecture to allow extensibility for other users to test different services for UAS in UTM.
    2. Built a simulation framework utilizing Airsim, ROS, and Unreal Engine for UTM applications for safe testing.
    3. Evaluated the simulation framework by testing it with multi-agent path-finding and precision landing.
- Multi-Agent Path Finding Algorithm for UAS
    1. Developed a multi-agent path finding algorithm that can be applied for UTM and xTM applications.
    2. Simulated and evaluated the performances of the algorithm utilizing Monte-Carlo techniques and applied into the High Fidelity simulation.
- Guidance and tracking law for quadcopters utilizing computer vision
    1. Developed a Linear Quadratic Gaussian (LQG) tracking law for quadcopters to track a visual target for precision landing.

2. Simulated the performance of the tracking law with shown results and comparisons to PX4's cascaded control law.

1.5 Thesis Organization

In Chapter 2, background and literature reviews are shared on the notional architecture of UTM and xTM, path-finding algorithms, as well as precision tracking and landing control methods utilizing computer vision. Chapter 3 discusses the framework conducted for the design and architecture of the High Fidelity simulation framework and its applications in the next two chapters. Chapter 4 shares the path-finding strategy proposed for multi-UAS operations in UTM. Tests are conducted utilizing a low-fidelity Monte Carlo and the High Fidelity simulation built in Chapter 3. Chapter 5 presents the control law and tracking law for a UAS to track fiducial tags and is tested in the High Fidelity simulation.

CHAPTER 2

LITERATURE REVIEW

2.1 Unmanned Traffic Management

2.1.1 UTM Requirements from FAA

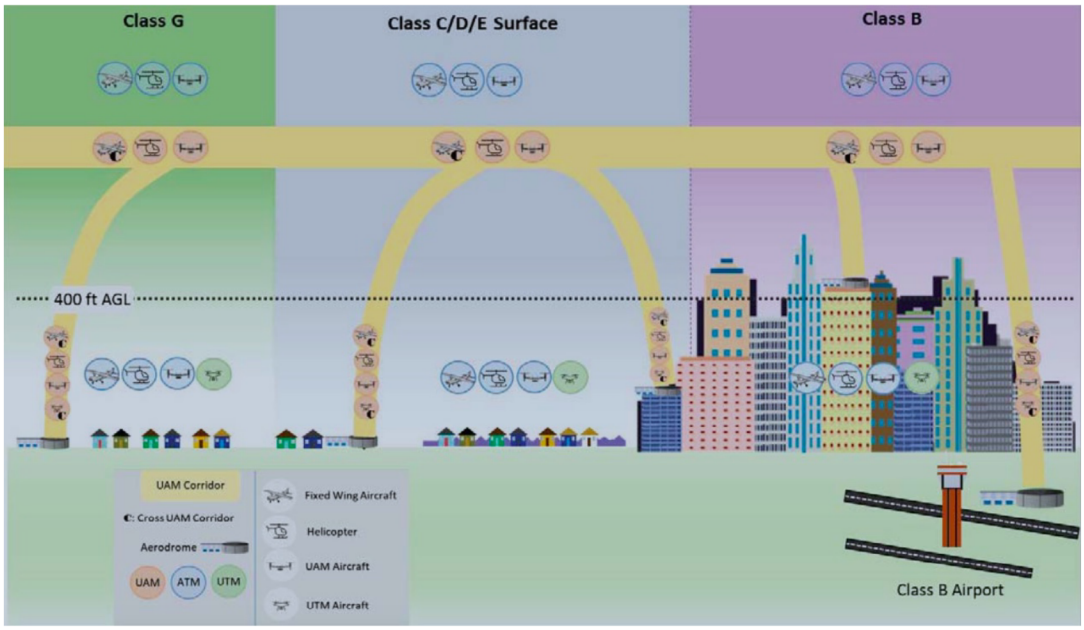


Figure 1: Diagram of UTM Operations in Context of Airspace classes [1]

In the FAA's ConOps of the UTM system, they present operational specifications that the UTM should be able to serve operations below 400 feet above ground level (AGL). In addition, the FAA has defined what capabilities the UTM should be able to support. Requirements are as follows:

- The UTM will support operations for Unmanned Air Systems (UAS) in low altitude

airspace, particularly for traffic management.

- There should be multiple layers of information sharing and data exchange from operator to operator.
- Communication is conducted not through voice interaction but through a distributed information network.
- The UTM should include a set of federated services and a flexible framework that can manage multiple Unmanned Air System (UAS) operations.
- The framework should be flexible to support the evolution of the UTM for future required services and protocols.
- Most importantly, the notional architecture of the UTM encourages third party entities to support the FAA's UTM architecture by sharing information amongst one another as well as to the FAA.

These requirements from the FAA are important to take into account for the development of the simulation, particularly for third party entities, so that when live testing is conducted, correct protocols are already verified and validated during the simulation.

2.1.2 Universal Service Supplier

Figure 2, displays the notionally architecture of UTM. In Figure 2 the light blue lines represent the relational boundaries between the FAA and industry standards for the infrastructure, and services of the UTM system. In the notional architecture the Universal Service Supplier (USS) has three main roles:

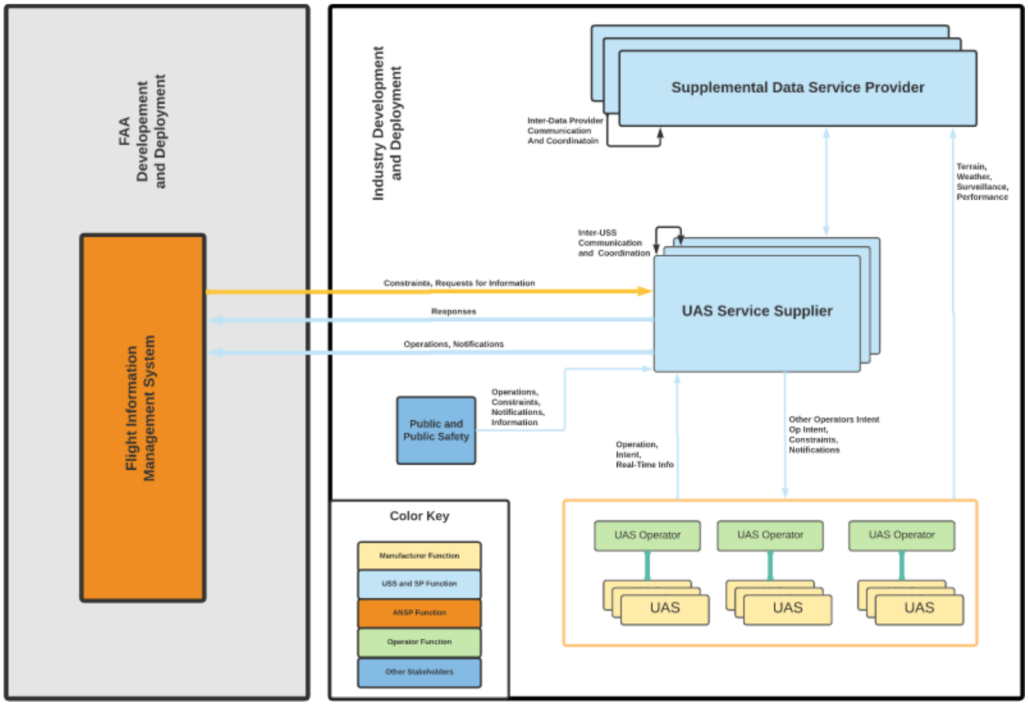


Figure 2: UTM Notional Architecture

1. It is a communications bridge between UTM to support Operators to meet regulations and protocols.

2. It provides the Operator with information about other planned operations around a volume of airspace to ensure safe mission conduction.

3. It keeps track of current and previous operations for record recall and information sharing.

In addition, the USS services provides operational planning, intent sharing, deconfliction, authorization of airspace, and various other services for future consideration. The USS plays an important role with the FAA since it will be be the main bridge port of

how the FAA receives information about air activity with UAS.

USS must also be able to provide strategic deconfliction of UAS to prevent fatal injuries from occurring due to airborne collisions [23]. NASA has specified that the USS must follow similar protocols of ICA DOC 9854, in regards to ”Global Air Traffic Management Operation Concept”. In this there are three layers of conflict management:

1. Strategic Conflict Management, which applies strategic actions prior to departure and deployment of UAS

2. Separation Provision, which is the process of keeping aircraft with a specified separation criterion, (a avoidance radius or collision bubble).

3. Collision Avoidance, which is activated when Separation Provision Layer has been compromised.

In chapter 4 when the path-finding algorithm is introduced, the first two items are analyzed since the path-finding algorithm proposed is conducted prior of UAS takeoff and also considers the collision radius for the respective UAS.

2.2 Path-Finding Algorithms

2.2.1 A* and Applications for UAS

A* is a popular path searching algorithm, due to its efficiency and adaptability to be used to solve various path-optimizing problems [30]. This is because A* utilizes a heuristic, a rule of thumb, to conduct its evaluation on searching for the next node to visit during its search to find the the optimal goal path.

$$f(x) = g(x) + h(x) \tag{2.1}$$

From Equation 2.1, $f(x)$ is the evaluation of the *total cost* to traverse from start to node x, $g(x)$ is the cost to go from start to node x. Finally $h(x)$ is the heuristic cost evaluation, and can be adjusted depending on the problem that must be solved. Typical heuristics used are the euclidean distance, where $h(x)$ is the straight line distance from the current node x to the end goal. The other commonly used heuristic is the Manhattan distance which is the evaluation from the current node x to the end goal node utilizing a "taxicab geometry". It must be noted that for A* to guarantee an optimal solution the heuristic evaluation must be admissible, that is the heuristic must *underestimate* the value, otherwise possible solutions will seem worse then they are and will not be considered an optimal option [13].

The A* algorithm incorporates both $open$ and $closed$ sets. The $open$ set are the potential $frontier$ nodes that can be visited, while the $closed$ set are nodes that have already been traversed. The $open$ set is typically utilized with a *Priority Queue* an abstract data-type where the nodes that have the lowest f value are placed at the top of the list. A* then "pops" off this node and conduct its evaluation, until it reaches the goal.

For UAS applications a study from [42] compared the performance of A* to Rapidly-Exploring Random Tree (RRT) in various 3D UAS scenarios. To briefly explain RRT, the algorithm "shoots" random seeds or areas around the initial node, after this a new node is selected for that is nearest to the current node. If the random generated nodes cause collision then that node is not a possibility. This iteration continues until the goal is

reached. During this study, RRT and its variant Multiple RRT or MRRT was compared. MRRT's framework is the same as RRT but now multiple initial searches are conducted and attempts to connect other search nodes with one another. In this study a smoothing algorithm was then applied to A*, RRT, and MRRT to compare the quality of the solution as well as time complexity, the time to find the solution, and space complexity, the amount of iterations and the search space size. Results from this study revealed that A* had a better time complexity, lower space complexity, and the quality of the solution were better than RRT and MRRT.

2.2.2 Multi-Agent Path Finding Algorithms

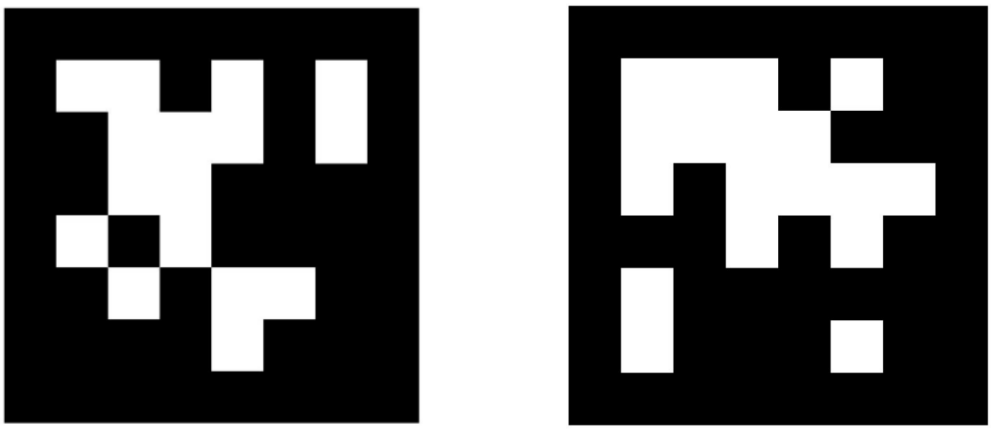
Multi Agent Path Finding algorithms are typically categorized in two ways: centralized and decentralized. With a centralized method, agents are assigned a designated path from a central planner during operations. With a decentralized method, it is up to the agents to search for a path independently of one another. Although these decentralized methods can provide scalibility, it also has multiple issues such as the difficulty of attempting to minimize a global cost function, as well as having situations where conflicting path plans are given due to constant changing constraints from individual agents [8]. In addition, with a decentralized method, communication with manned aircraft in the shared airspace would be difficult to integrate. Last, implementing the *same* decentralized algorithm for different and various UAS that belong to different third party industries is not feasible for application. It would be considered impractical to demand competitive industries to implement the same algorithm.

Centralized methods are favored for the implementation of UTM, since it would provide a centralized planner done by one or multiple USS, where shared intent is communicated throughout the network. In addition, this allows abstraction of the issue of path-planning of UAS with different configurations and are controlled by different types of Operators. Conflict Based Search or CBS, is centralized method that searches for any possible conflicts, utilizing a *Conflict Tree*. Although this is a favorable, CBS fails to do well in open-grid spaces, in comparison to A* [34], which in UAS operations is not desired. Techniques that apply A* such as Hierarchical Cooperative A* (HCA*), Cooperative A* (CA*) and Hierarchical Path-Finding A* (HPA*), all utilize A* as the main framework for path planning, implementing a ”divide and conquer” methodology of breaking the grid into abstract graphs to reduce space complexity [35], [5] of the problem. In addition, these algorithms utilize a reservation table to reserve waypoints for planning of other agents.

2.3 Precision Tracking and Landing

2.3.1 AprilTags

AprilTags are heavily used by the robotics community for local relative pose detection and estimation. These markers are designed for detection by computer vision systems [26]. The AprilTag provides a 6 Degree of Freedom (DOF) localization of the pose orientation once the computer vision detects the image, providing both a robust and quick detection of the system. AprilTags are open licensed and in comparison to the ARToolkit, the computational cost to detect the fiducial tag is lower than its predecessor.



(a) Tag ID 0 (b) Tag ID 4

Figure 3: AprilTag family 36h11

Figure 3 displays two configurations of the AprilTag family 36h11, where the 36 represents the number of pixels of the image class and the h11 represents the hamming distance of the tag family for error correction. The larger the hamming distance value, the lower the false detection or error detection of the tag.

2.3.2 Linear Quadratic Regulator

Linear Quadratic Controllers (LQR) are a widely used and popular control design for aerospace applications. This is due its outstanding performances and robustness to disturbances input into the system. The whole application of LQR is conducted by minimizing a cost function of J, between the state matrix, Q, and actuator matrix, R [20]. Reason for implementing a linear quadratic controller were due to the studies from [38] and [32], where in both studies, results showed a stable response from LQR in reduction

of overshoot from step inputs to a quadcopter system. In addition, LQR provides the advantage of the reduction of tuning multiple gains from a nested controller requiring the user to only define the Q and R values to the system to optimize the cost function J.

2.3.3 Kalman Filter

The Kalman Filter, or Linear Quadratic Estimation, LQE, is a popular state estimation technique used in aerospace as well as autonomous control and guidance of vehicles. The state estimation technique was developed by Rudolf E. Kalman, and is utilized for extraction of information from noisy data of [28]. It is typically misunderstood that a Kalman Filter is only suited for Gaussian noise, when in fact it can be applied to white noise as well. Kalman Filter estimates are done by fusing sensor information with a model as reference, in addition what makes a Kalman Filter extremely power is that one does need a sensor to estimate a particular state. For instance if there is only a velocity sensor in the system but position needs to be tracked, position can still be found because of the model used in the Kalman Filter.

Kalman Filters have been applied for computer vision tracking as well, in a study from [11], an experiment was conducted with a surveillance camera to track a visual moving target utilizing a constant velocity model. To showcase the immense popularity of the applications of Kalman Filters [6] an overall study of the applications and research topics related to Kalman Filters showcases an exponential growth of the topic, with one of the main growing topics for the application of Kalman Filters being object tracking and

leader-follower systems. For tracking with computer vision most studies utilize a state-transition model (which will be discussed later) for constant velocities as seen, in studies from [39] and [4], where in these studies tracking is done of pedestrians and moving vehicles from a surveillance camera.

2.3.4 Precision Tracking and Landing with Apriltags

Precision landing and tracking for autonomous UAS is a popular topic, since it allows UAS to conduct precise and safe landing protocols in GPS denied environments. A study from [12] conducted simulation tests for precision landing in a commercial poultry house utilizing a mobile GPS signal to ping its location to the UAS, landing on a mobile fiducial tag, and landing at a fixed fiducial tag, with minimal errors. However the tests were at very low altitudes at 3m, with no disturbances to test the robustness of precision landing of the system, in addition the speeds of the quadcopter was limited to 1m/s, which is extremely slow for tracking. Another study from [21] developed a UAS with a Proportional Integral Derivation (PID) controller to track an Unmanned Ground Vehicle (UGV), utilizing Apriltags and an added kernel correlation filter to remove any problems of occlusions, where one edge of the Apriltag is blocked from view, preventing vision tracking. Although results proved favorable, in the study whenever the UAS loses track of the target, it will stop tracking and hover in place until the tag is found again or if the UGV has found the target.

CHAPTER 3

SIMULATION DEVELOPMENT FOR UTM OPERATIONS

3.1 Introduction

With the technological advances of CPU and GPU for computers, simulations are leveraged for the testing of autonomous systems. With simulations it provides users with the capabilities to understand more about the environment simulated as well as address if there are any organizational changes that need to be applied [15]. For autonomous vehicles *CARLA* is a popular open source simulation framework for testing these grounded vehicles [9]. For UAS, a simulation framework for UTM operations is crucial too. This would allow future users the ability to test out new planning USS, algorithms, and information trading protocols prior to live testing. In this chapter, implementation of a simulation framework that follows the UTM notional architecture is introduced and discussion of the derived simulation architecture is described. Application is briefly discussed and implementation with path planning and guidance testing are shown in later chapters.

3.2 Related Simulation Work

The majority of open-source simulation environments for UTM applications utilized Robot Operating System (ROS) as part of the framework for the simulation. One example is a simulation developed utilizing ROS and Gazebo [24], [25]. In these framewors the simulation was able to provide the following:

- Utilize U-space architecture for UTM management of drones.
- Provide framework for multiple types of UAS (fixed-wing and quadrotors), as well as pre-flight and in-flight services.

However there were some issues with the simulator:

- The framework did not provide user friendly access for additional services to be included in its environment.
- The framework is tailored towards U-Space operations of the system, which is Europe's current implementation of UTM.

Other simulation environments outside the scope of ROS and Gazebo is *BlueSky*. *BlueSky* is an air traffic management simulation utilizing Python as the main source code [14]. In the scope of this simulation, *BlueSky* provides the following:

- Open source development for future applications.
- The software is free without need of expensive integration.

However *BlueSky* does have limitations in its simulation environment:

- It only provides services or simulations based on commercial sized aircraft modules.
- Operator interface to change or configure UAS flight trajectories is complicated.
- Real time physics based interaction of the simulator is not incorporated.

The initial framework developed utilizing the ROS and Gazebo provides a desirable simulation environment allowing real time physics as well as the possibility of incorporating SITL (Software In The Loop) flight controllers such as PX4 and ArduPilot. Although the the simulation environment from *BlueSky* does not implement real-time physics it does provide low overhead estimates of mission success, allowing iterative testing of situations.

3.3 Previous Work and Motivation to Change Platform

Gazebo is an open source 3D simulator that supports the development of single and multi-agent operational environments [18]. It provides 3D graphical representation of the agent(s) while also providing sensor plugins for feedback of the engine. Because of the open-source capabilities of Gazebo, it allows custom plugins and robot models to be built and added on as a feature for the Gazebo environment. This allows the user to design and configure any situation needed in real world environment. Gazebo in addition has Software In The Loop (SITL) plugins for flight controllers such as PX4 and Ardupilot. This allows a *High Fidelity* simulation to be implemented, that is provide capability to simulate UAS in 6 Degrees of Freedom and provide interactions with other agents or systems.

During the initial design of the simulationt, Gazebo was utilized for the testing of the Third Party Landing Service and had successful results, however there are some notable limitations. One limitation is the resolution fidelity of Gazebo, this would limit any photo realistic environments for testing of other third party services implementing computer vision as an additional feature (such as surveillance of fire, dropping care packages

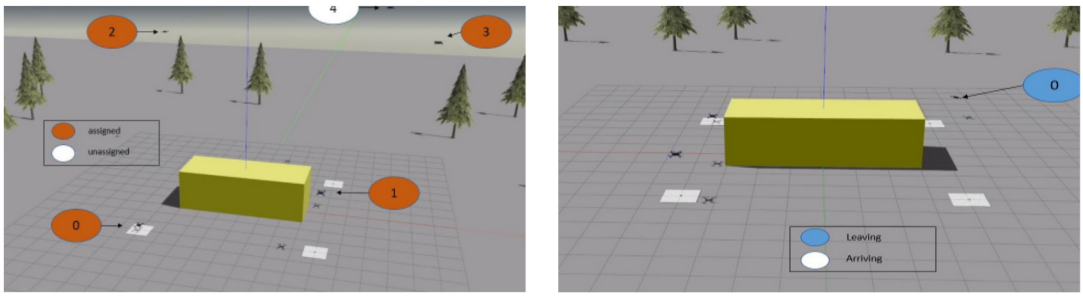


Figure 4: Gazebo UTM Simulation Initially Implemented

based on human recognition, etc.). In addition, the real time run factor of the simulations conducted was 10 times slower than the actual real application of the simulation. That is when the simulation would take 20 minutes to complete, the real life implementation of guidance and control of the UAS would take 2 minutes.

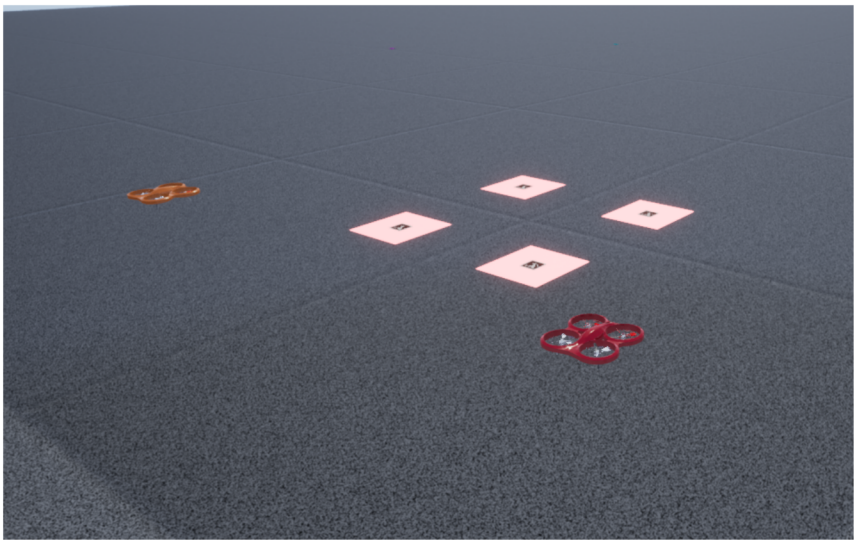


Figure 5: Airsim Simulation

Implementation of a higher fidelity simulation has moved to Unreal Engine with Microsoft's Airsim. Airsim is Microsoft's simulator for quad-copters and cars that is built

on Unreal Engine, which is typically utilized for artificial intelligence such as deep learning and computer vision applications for autonomous vehicles [33]. Airsim also has PX4 and ArduPilot Software In the Loop (SITL) and Hardware In the Loop (HITL) support for its simulations. This allows the simulation to be just as realistic on the software side as the environment. Furthermore, Airsim provides integration with ROS, thus allowing the transition from Gazebo to Unreal to be easier to migrate to. From this, requirements are defined for the development of the UTM simulation:

1. The simulation must follow communication protocols of the notional UTM architecture discussed in 2.1.

2. The simulation must have capability to implement flight controllers and provide high fidelity visualization of the environment.

3. The simulation and its structure must provide extensibility for future UTM applications and for developers.

The following sections will discuss the software architecture of the development of this simulation and the framework implementation from this derived architecture to meet these requirements specified.

3.4 Software Architecture of High Fidelity Simulation

The purpose of developing this software architecture was to provide a road map on the implementation of the code and how it falls within the scope of the user using interacting with this simulation. This allows ease of integration for future work and additions

to the simulation environment where users can extend code from given Automatic Programming Interfaces (API) provided or extend the base. The software architecture was implemented utilizing the a Context, Container, and Component (C3) Diagram, which is variation of the C4 Diagram. This diagram provides a scope of the software system, with the container providing the high level scope of how the user will interact with the software, the container represents the applications that will be used for the development of this system, and the component which is the classes,code, and API implemented in the system.

3.4.1 Context: The Buisiness Logic

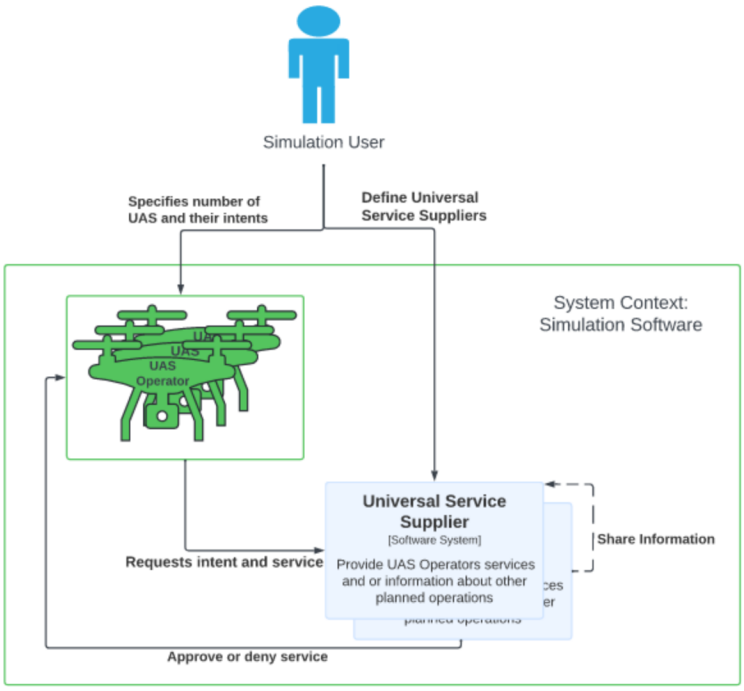


Figure 6: Context Diagram

Figure 6, displays the context diagram. This diagram provides the developers

the high level overview of what the simulator user should be able to do with the given simulation. To be able to do this the buisiness logic of UTM should be understood. Once this is understood, the simulation software should provide the user the ability to specify the number of UAS operators and what their intents are. In addition, the simulation user should be able to specify what kind of USS should be run in the simulation, as well as allowing the user to add in any additional USS. Once specified the UAS Operators and the USS will then conduct their simulations and the user will be able to see the operations.

3.4.2 Container: What's in the System

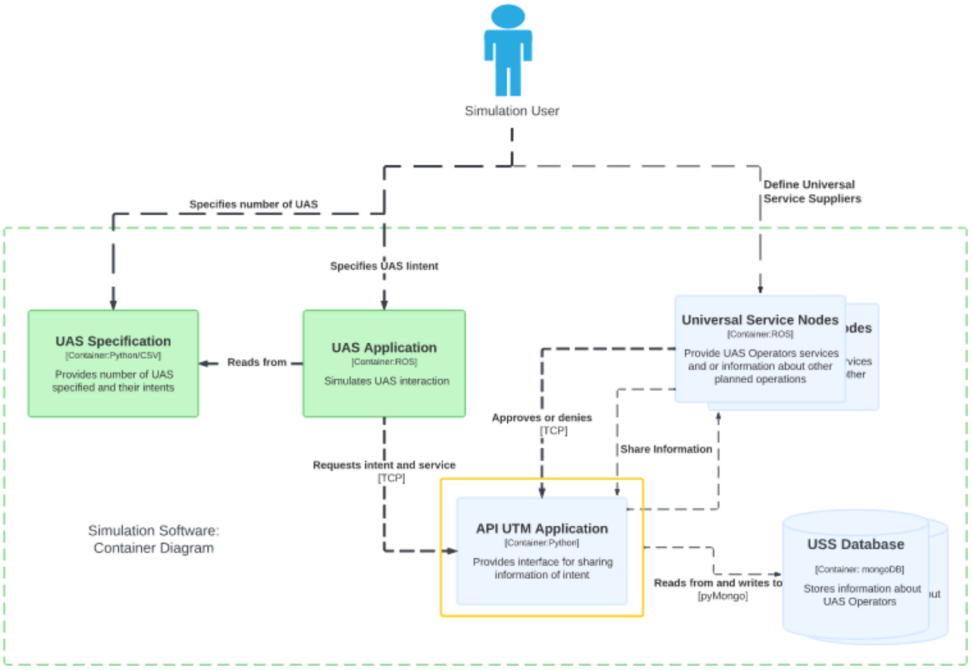


Figure 7: Container Diagram

The container diagram can be considered a "zoomed" in scope of the software system, which for this application is shown in Figure 7. For the specifications of the

UAS Operators a *UAS Specification* container specified from a CSV file is defined by the user. From this, the UAS intents are defined by the *UAS Applications* container with the specifications read from the CSV file. During this simulation, the UAS Application then utilizes the *API UTM Application* which acts as an interface for the communication protocols for UTM operations. For the USS, the *USS Node* containers also calls the API UTM Application to search for incoming UAS that are in need of its respective service and approve or disapprove the service. These UAS are then written to the USS Nodes respective database. In addition, USS share their information from this API as well. By leveraging this API UTM Application as the interface, it prevents tight coupling from occurring. This provides extensibility for future USS and or unique UAS because all they need to do is to use this interface for communications with one another.

3.4.3 Component: What Builds the Containers

The component diagram is a further ”zoomed” in view of any specific applications or interfaces. From the API UTM Application, this container consists of a Database Controller, which is a subset of classes for the interface of querying database protocols and sharing of information. As previously stated this prevents the tight coupling and tight dependencies between UAS and USS, decoupling the design and allows easier development. With the skeleton of the simulation architecture shown, the actual application and integration of the frameworks, or tools, using ROS and Unreal Engine can finally be discussed.

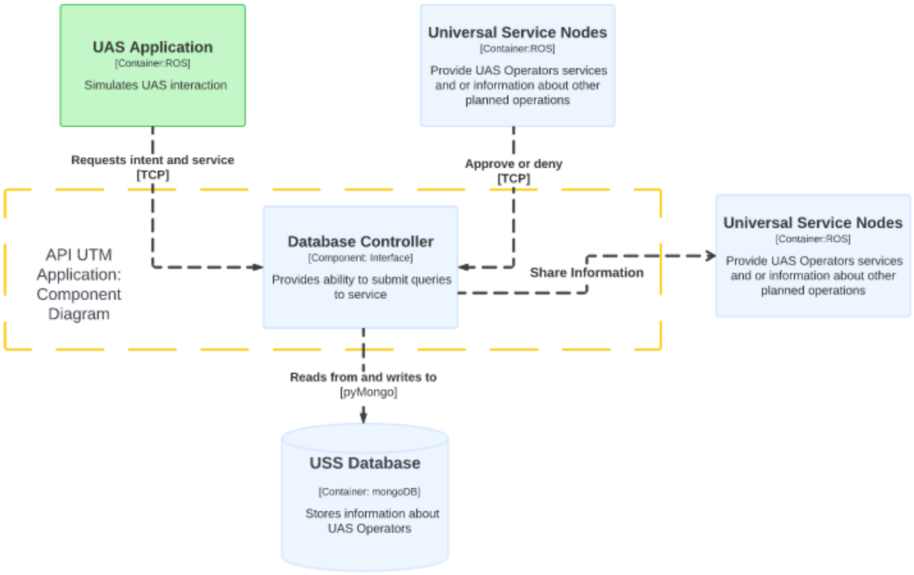


Figure 8: Component Diagram

3.5 Implementation of Software with Frameworks

As discussed in Section 3.3, the High Fidelity simulation visualization was changed from Gazebo to Unreal Engine. This section discusses the integration of how the frameworks work together with consideration of the software architecture. In addition methods of integrating the notional Unmanned Traffic Management architecture.

Figure 9, showcases the high level infrastructure of the High Fidelity simulation. The simulation framework consists of Windows Operating System running *Unreal Engine* and Microsoft's *Airsim*. A virtual machine or Windows Subsystem for Linux (WSL) was then utilized to run *ROS*, *MongoDB*, and the *flight controller*.

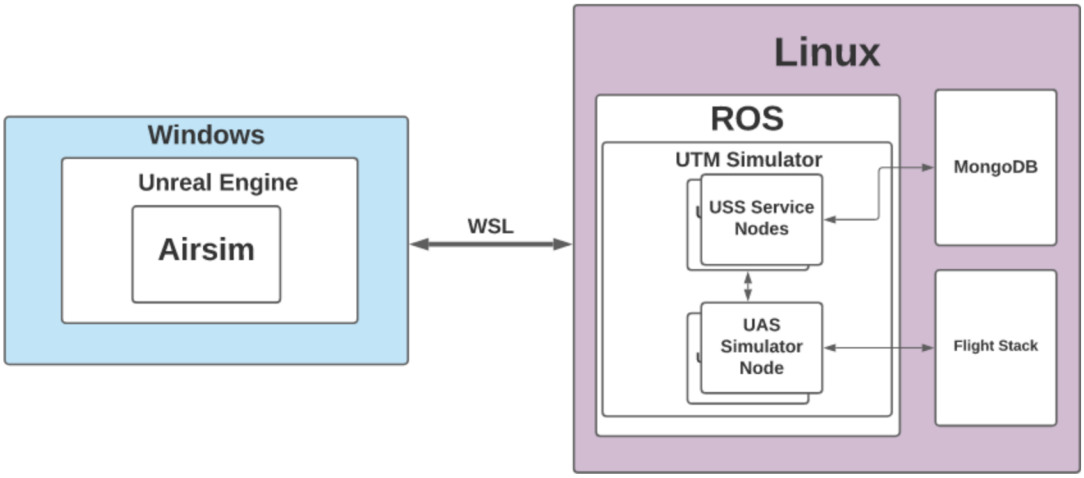


Figure 9: High Fidelity Simulation Framework

3.6 Back-End Framework

3.6.1 Robot Operating System (ROS)

Robot Operating System, or ROS, is a free and open-source middle ware communication system for the use and applications of single or multiple agents [29]. In addition ROS provides a system of peer to peer connection, is multi-lingual for faster connection types, and has a collaborative community for additional packages to be installed for implementation or further development. The main driving concepts of ROS implementation are the the usage of nodes, messages, topics, and services. Nodes are modules that perform some sort of computation such as path finding or computer vision feature recognition. These nodes can send out messages through a communication protocol known as a topic by publishing the topic at hand. By publishing this message other nodes can retrieve this information by subscribing to the topic name for its own purposes. ROS was utilized for the back-end of the architecture due to its versatility in allowing the user to implement

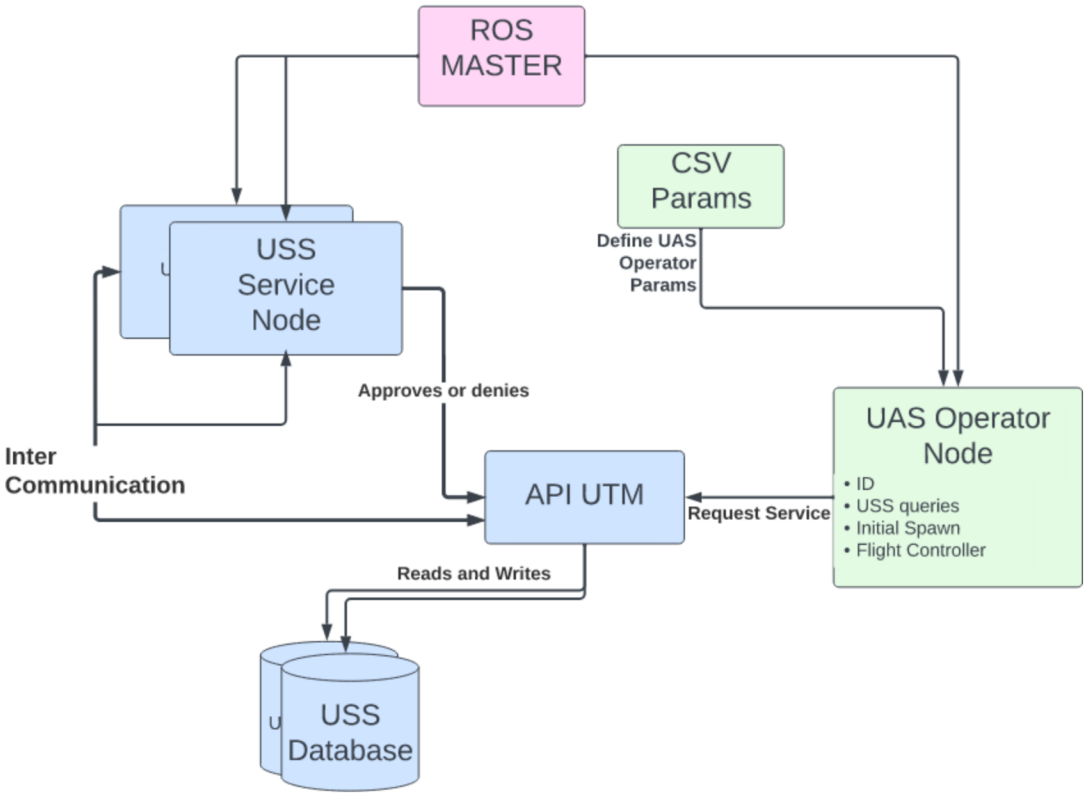


Figure 10: Architecture of Back-End Framework

code in either C++ or Python. In addition leveraging the publisher subscriber architecture was considered very favorable to simulate multi-UAS operations during the simulations, since the UAS ID can be included into the topic name. This allows communication to be shared between the correct UAS during simulation environment testing. Last, ROS allows ease and integration of different packages and modules for extensibility of future work, allowing modularity in the design of the UTM simulator. This provides additional USS packages or nodes that can be included into the framework of the simulation.

3.6.2 CSV Params

The CSV Params is a simple csv file where the user can define various parameters of the UAS, such as ID number, location of spawn, initial target location, and what kind of service or operation the UAS will conduct. This allows the user to run various different configurations of UAS and layouts without the need to hard-set parameters in the code base itself.

3.6.3 ROS Master

The ROS Master is a Python script that launches multiple UAS nodes based on the configurations defined by the *CSV Params*. The ROS Master is needed in ROS 1 to allow communications of the respective publisher and subscribers to communicate to one another, without it, the simulation will not be conducted.

3.6.4 UAS Operator Node

The UAS Operator is the owner of the individual UAS. In this simulation environment, the UAS Operator sends the UAS to a relative waypoint, and shares information intent to the data service provider and the respective service requested.

The UAS will have an identification number attached to whatever its topic name is to allow proper communication protocols to utilizing the ROS framework. In addition this UAS will either run PX4 SITL or Airsim's SimpleFlight flight controller, and is a group 1 UAS quadcopter.

3.6.5 Universal Service Supplier Node

The Universal Service Supplier (USS) Node is an abstraction of the actual USS in UTM. A USS node provides a service to any UAS that is requesting this node. Each USS node has a mongoDB collection that, where the node utilizes API calls for any incoming queries of UAS requesting to use its service. If so, the USS conducts its protocols and procedures and approves or disapproves it to the UAS. These USS nodes can also communicate to each to simulate the Inter-USS communication protocol with the API.

3.7 Front-End Framework

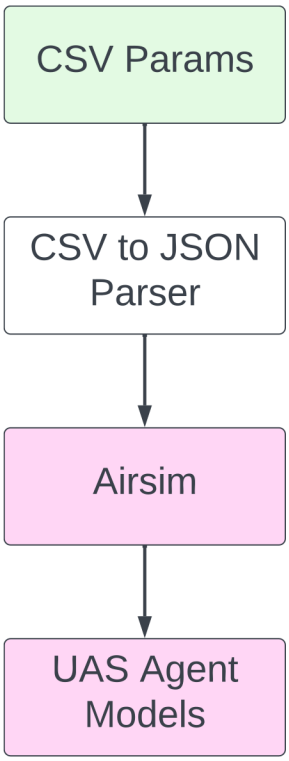


Figure 11: Architecture of Front-End Framework

For the front-end of the framework, the same CSV Params was utilized, where a Python script was used to convert the parameters from the CSV to JavaScript Object Notation (JSON) format and set to the Airsim's JSON *settings.json* file. Parameters such as UAS model type, flight controller type, spawn location, and additional sensors are specified to the simulated respective UAS. Once set, the user can then begin running the ROS Master node to connect the UAS agent models to simulate any situation they desired.

3.8 Results

The following chapters provide integration of this High Fidelity simulation to test the interaction of the services and operations. Figure 12 showcases the implementation of the High Fidelity Simulation. In Figure 12a 50 UAS are spawned from the specifications of the CSV Params, Figure 12b is where the UAS conduct precision landing utilizing fiducial markers, and Figure 12c is the application of the High Fidelity simulation for a path planning USS. The High Fidelity simulations were operated utilizing a AMD Ryzen 7 processor with 5800H, 32GB of RAM and NVIDIA GeForce RTX 3060 Ti graphics card.

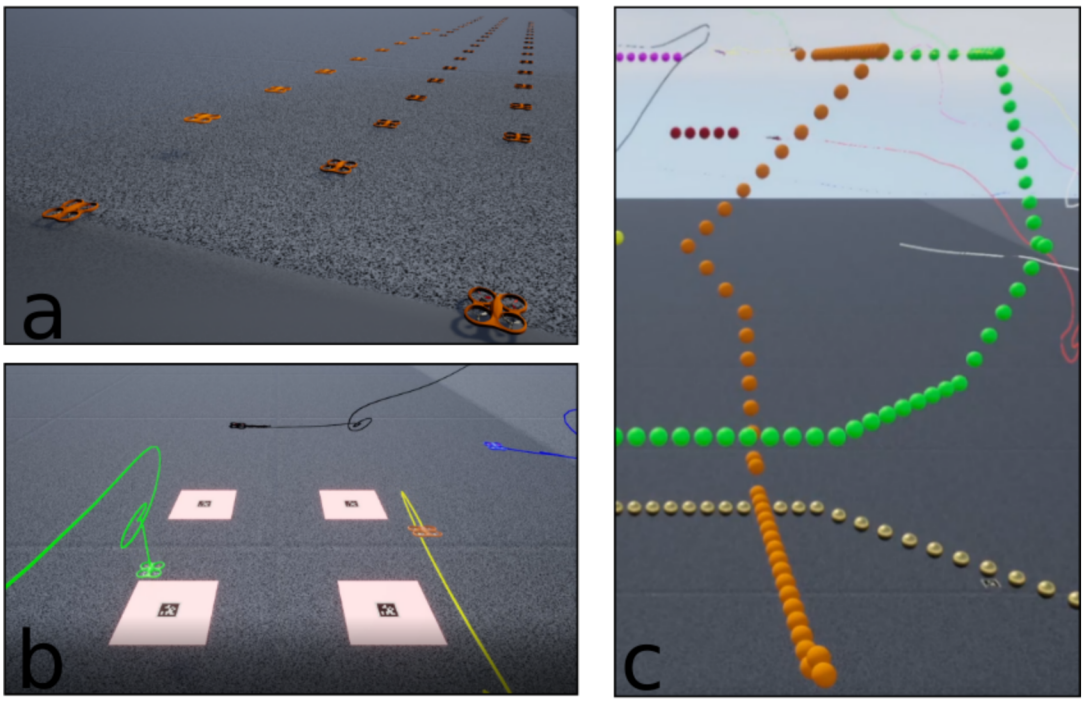


Figure 12: Utilizing the High Fidelity Simulation

CHAPTER 4

MULTI-AGENT PATH FINDING ALGORITHM FOR UTM OPERATIONS

4.1 Introduction

For safe operations to be conducted, one of the key principles is that Unmanned Air Systems (UAS) must be able to avoid each other, and that UAS operators should have complete awareness of all constraints in the airspace [23]. To implement this in UTM, the UAS Operator would submit the intended flight trajectory to a Universal Service Supplier (USS). The USS would then aid in defining the operation volume for the UAS, by considering other UAS Operators' flight intent. This protocol and implementation with a USS provides *strategic deconfliction* between all UAS operators.

Although there are numerous papers that implement real time path finding solutions during in-flight operations of UAS [41], [31], [22], there are few path finding methods implemented *prior* of UAS takeoff. A centralized multi-UAS path finding method, prior to takeoff, is proposed that implements a hierarchical approach. This implementation is adaptable for different configuration layouts and can be easily integrated towards the UTM architecture proposed by the FAA. The algorithm is modular and allows for variation of heuristics or algorithms to be applied, particularly for the low level search. This algorithm is tested numerous times with Monte Carlo simulations where between 10 to 100 UAS of type quadcopter are requesting a path to be planned from their start and goal points respectively within a defined air volume density of 100 x 100 x 75. Last, the

High Fidelity simulation environment, is incorporated into the algorithm implementing UTM and is tested.

4.2 Motivation and Requirements

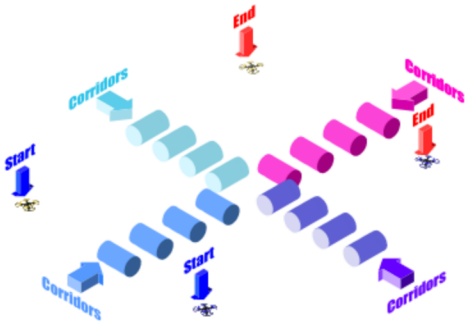
In UTM, it is integral that USS provides support in deconfliction of flight trajectories for UAS operators [23]. It is crucial that the USS identifies all constraints within the sector and provide safety mitigation as well as support for UAS operations. Because of these requirements, centralized planning for multiple UAS would be the best approach to keep track of all knowledge of UAS operations within the vicinity as well as any restricted areas.

It is desired that protocols of travel for UAS follow similar protocols of general aviation such as the usage of *airways* and *corridors* for future extension of UTM with an extensible Traffic Management System, xTM proposed by NASA. Where in the future, instead of segregated airspace from UAS and manned aircraft, the two will be integrated together [16]. To do so, consideration of how manned aircraft must traverse should be taken into account. Consider the following example of a pilot planning a flight from Kansas City, Missouri to Gainesville, Florida.

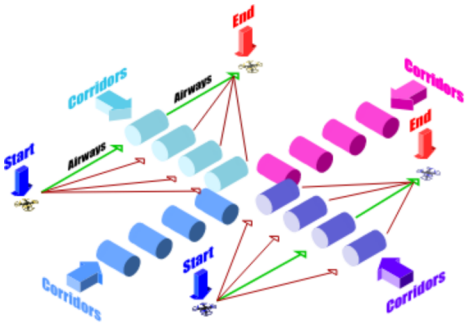
1. The pilot determines what airways that are to be traversed in each region or sector of a global map.

2. For high density airspace, particularly for Class B, the pilot must traverse through a specified *corridor* to mitigate any risks of collisions or accidents from occurring.

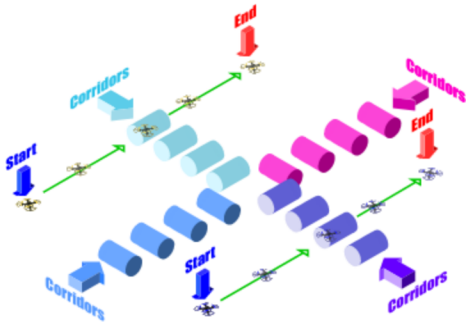
3. This flight trajectory must be shared to Air Traffic Control (ATC) prior to flight and during flight.



(a) Define Start and End Destinations



(b) High Level Search of Corridors by Searching Airways



(c) Low Level Search to get to Corridor

Figure 13: Path Finding Strategy for UAS

From this method the initial start and end points are defined so that they can be connected to these points of their respective *airways* and *corridors*. This allows the ability to define an abstraction of the path the UAS must transverse across each region in the airspace. This protocol done by pilots would be favorable to use for path finding of UAS because of the following:

1. By finding abstract waypoints it reduces the search space, or the number of iterations to find the goal destination.

2. For each abstract waypoint do a low level search to find an optimal path to be to get to the desired location, utilizing any search-algorithm desired.

3. The ability to keep track of flight trajectories for UTM protocols and allow flight intent to be shared to other UAS Operators and USS.

4. Mitigate any risks prior to flight by considering current paths taken by other UAS operators.

5. Easier integration and compatibility with future extension of UTM or xTM for flight operations with both manned and unmanned aircraft due to this shared intent from UAS systems.

4.3 Method Implementation of Algorithm

The main procedures of this algorithm are as follows:

1. Define the *map* and divide the map into *regions*. In the demonstration and simulation setup there is a 100 x 100 x 75 map. This map is to then divided into 4 regions of 25 x 25 x 75.

2. Determine the boundaries of the regions that can connect to other regions as *entrances*, or *adjacent entrances*, and define *corridors* that link these adjacent entrances to each other.

3. Define a unit cost to travel from one end to the other end of the corridor, and cache the cost of each corridor. Look inside of each regions' entrances, or *inner entrances*, and define *airways* by connecting each inner entrance to another inner entrance.

4. Incoming UAS queries for waypoints are then to be received, and a *High Level Search* is planned for UAS with consideration of waypoints in the *reservation table*. For the current scope of this work, UAS are group 1 quadcopters, with same flight characteristics.

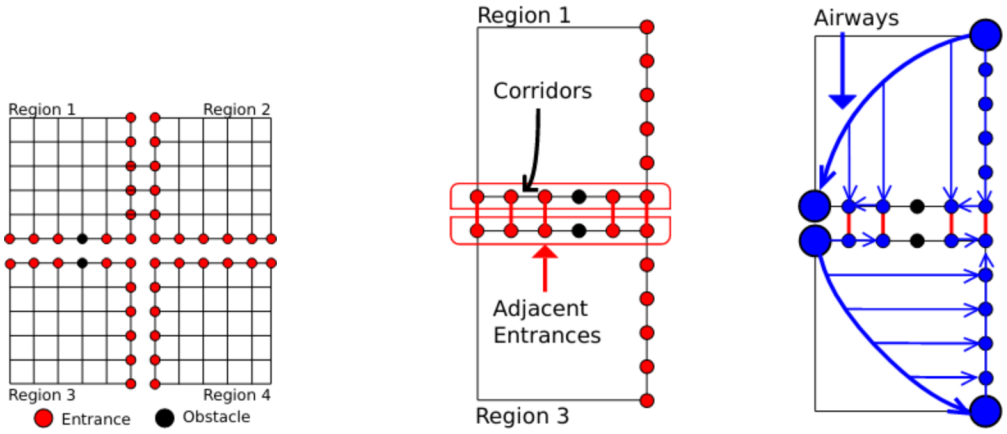
5. With these high level waypoints, conduct an iterative *Low Level Search* to get from one high level waypoint to the next high level waypoint, until the goal is reached.

6. Once the path is planned cache these waypoints in to the *reservation table*, when the UAS has reached its end destination, the respective waypoints are removed from the *reservation table*.

For the remainder of this section, each of the components and steps of the algorithm is discussed.

4.3.1 Maps and Regions

The map can be considered as the topological predefined space. In this map there are regions, which are sub-components of the whole map. The 100 x 100 x 75 map is segmented into 4 regions as shown in Figure 14a. It must be noted that there are z number of regions overlayed on top of each other, which is the constrained by the height of the map. For this case 15 layers of these regions are specified. Afterwards it is up to the



(a) Dividing the Map into Regions and Defining Entrances

(b) Connecting Adjacent Entrances

(c) Airways for one inner entrance

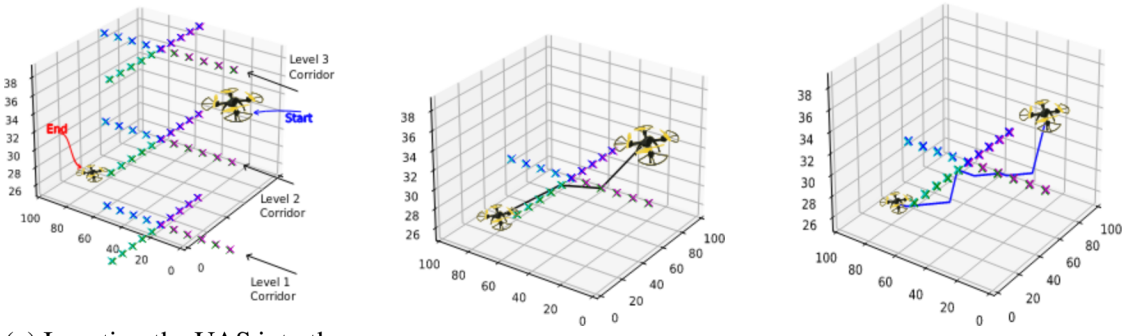
Figure 14: Abstraction Process

user to determine and specify possible entrances on the outer grid of each region based on the parameter of the user, for this case an entrance every 5 units, as long as an obstacle does not occupy the respective location. It must be noted that the map and regions have no knowledge of any agents in the vicinity, and only knowledge of adjacent regions and defined static obstacles.

4.3.2 Build Corridors and Airways, Not Walls

To build corridors, search for possible adjacent entrances (Figure 14b). To do so, look at each entrance of the region and determine if the entrance has an adjacent entrance that is not of the same region, if so this is a possible corridor path. Once completed, the pairs of adjacent entrances are by used to define the multiple corridors. Afterwards, set a uniform cost to traverse the corridors, for this case a unit cost of 1 and cache the corridors with their entrances and costs.

Once all corridors are defined, begin building the airways (Figure 14c). This is conducted by looking for entrances that belong to that respective region, which will be notated as inner-entrances. The user then connects each of these inner-entrances to each other and compute the cost to traverse from one inner-entrance to the other inner-entrance. For the cost compute the euclidean distance between the two inner-entrances, but using a simple A* to compute the cost could also be another possibility. Each of these connections between the inner-entrances are considered an airway for a UAS to traverse to. With the airways and corridors built and defined, the method is ready to service incoming UAS that need a path planned.



(a) Inserting the UAS into the Search Space

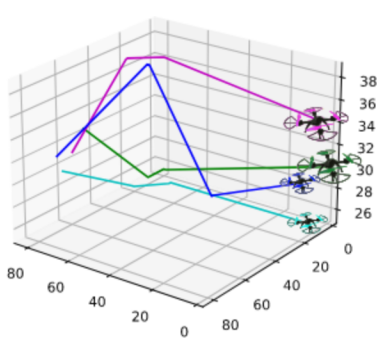
(b) High Level Search

(c) Low Level Search

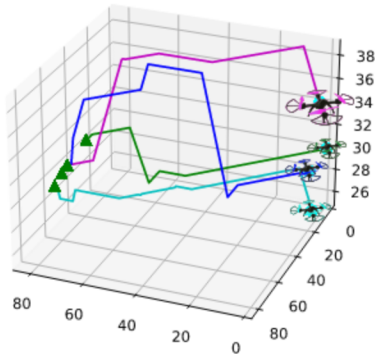
Figure 15: Single UAS Path Planning

4.3.3 Searching High and Low

Once a UAS has requested to find a path, begin a high level search by connecting the start and goal to their respective regions thus connecting them to all airways within the region, shown in Figure 15a and Figure 15b. Search for the minimum cost for the



(a) Multi-UAS High Level Path Planning



(b) Multi-UAS Low Level Path Planning

Figure 16: Multi-UAS Path Planning

start to traverse across each region to arrive to the goal, and store these entrance locations and their cost. These entrance locations become the abstract waypoints for the UAS to traverse.

It must be noted that since the agents can move in 3-dimensional aspect, the search depth in the high-level search can be operated in different heights of altitude. This provides the opportunity for the algorithm to search for paths at different altitudes, in case entrances at the current altitude are occupied by other UAS. Although this will reduce the quality of the solution, this will still provide the service to the UAS.

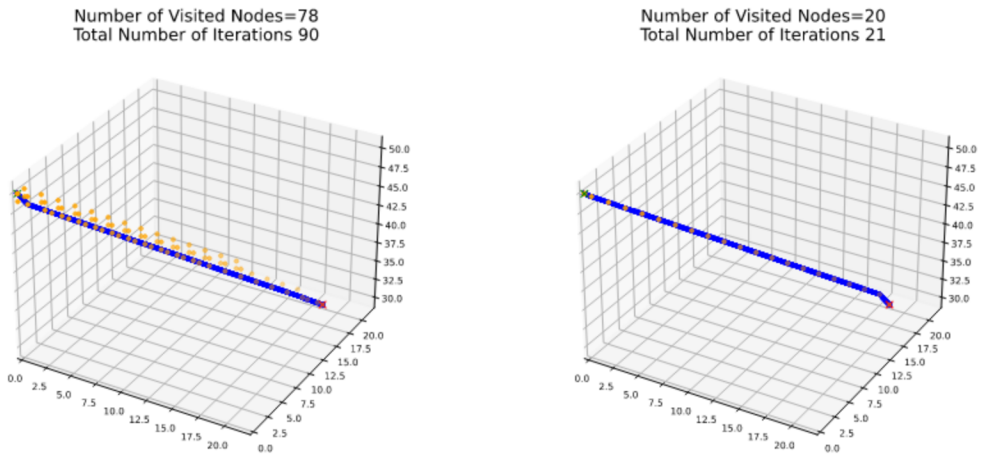
4.3.4 Low Level Weighted A*: Greed is Good

From the abstract waypoints, do an iterative low level search for each abstract waypoint utilizing a weighted A* implemented from [17] with the main equation in 4.1.

$$f(x) = g(x) + \epsilon h(x) + p \tag{4.1}$$

$$\epsilon = w\frac{h(x)}{g(x)} \tag{4.2}$$

With weighted A* the main driver is ϵ as shown in equation 4.2, essentially it is a ratio between the current heuristic cost $h(x)$ (euclidean distance from the current position to the goal position), and the current cost of traversal $g(x)$, multiplied by some constant w. Equation 4.2 can be seen as a ratio, the bigger $h(x)$ is compared to $g(x)$, weighted A* becomes a greedier-best first search method, from this amplification. Applying this weight is favorable particularly for the initial start of the search. This prunes, or removes, out search areas close to the starting location, and prevents the search space from going exponentially large due to the 3-dimensional space.



(a) Regular A*

(b) Weighted A*

Figure 17: Comparison Between Regular A* and Weighted A*

In addition, to prevent the weighted A* from overestimating, when $h(x)$ becomes

smaller, the ratio in equation 4.2 decreases as well, thus $g(x)$ has more weight place making the traversal much more conservative. To demonstrate and visualize the performance, Figure 17 shows the path planning between regular A* and Weighted A*, both with the same start and end locations. It can be seen that in Figure 17a has a search space that is almost 4 times larger compared to the implemented Weighted A* method. With complex paths, the regular A* would take longer to find a solution. Even though the solution from Figure 17b, does not yield an optimal solution, "close" or "good" solutions are fine, with a faster solution time in return.

Last, a static penalty cost, p was included for change of altitudes, that are scaled higher if the UAS has to move in the opposite height direction of the actual goal points. This would discourage diagonal movements, and encourages straight line maneuvers for UAS, whenever possible. Once a path is found from the abstract waypoint to the goal, the low level waypoints and an inflated radius, defined by the UAS size, is inserted to the reservation table.

4.3.5 Reservation Table: Sorry this Area Has Been Reserved

The reservation table caches all information of planned UAS flight trajectories. These cached paths are considered dynamic obstacles for incoming UAS who query for a path to be planned. By storing this information in the reservation table, new paths planned avoid possible intersections specified by a collision radius and location. In addition, for UTM applications these flight trajectories and paths can be shared to other USS and the FAA for knowledge of air density in the regions. Once the UAS Operator has reached the

endpoint, the path is then removed from the reservation table.

Table 1: Reservation Table for 3 UAS

UAS ID	Reserved Paths
UAS 1	wp_1, wp_2,...wp_n
UAS 2	wp_1, wp_2,...wp_n
UAS 3	

4.3.6 Priority Planning: Who Goes First?

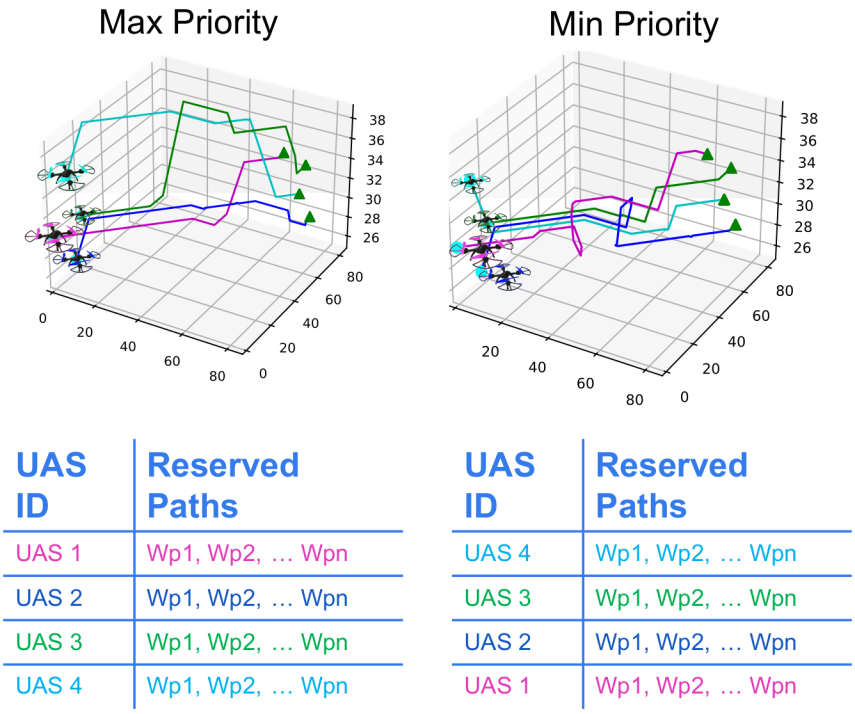


Figure 18: Priority Planning with Max and Min

It must be noted that because this planning protocol utilizes a centralized basis, priority of which UAS should be planned first matters in the overall solution given to

other UAS. In addition, this affects the overall success to be able to find a path for all UAS. Figure 18 showcases the different solutions based on the prioritization of UAS for both maximum distances and minimum distances respectively. That is with a maximum distance prioritization, UAS who have to traverse the farthest are planned first. With a minimum distance prioritization, UAS who have the shortest distance to traverse are planned first. The performances between the minimum, maximum, and unsorted methods will be evaluated to test which yields the best performance metrics.

4.4 Simulations and Results

In this section, discussion of the simulation setup and results are shared utilizing Monte Carlo and the High Fidelity simulation. In the Monte Carlo the algorithm is tested from 10 to 100 UAS quadcopters who at the same time instance request to have a path planned within a 100 x 100 x 75 airspace. Evaluation of the success rate, the time complexity, space complexity, and quality of solution were compared utilizing different prioritization strategies. Drawbacks of the algorithm are discussed and possible solutions to solve these issues are offered. In the High Fidelity simulations the algorithm is integrated as a USS supplier that provides path planning to UAS using Unreal Engine, Robot Operating System (ROS), and Microsoft's Airsim from the simulation framework in Chapter 3. High Fidelity simulations are then ran with 10 UAS for evaluation of the performance of the algorithm. The Monte Carlo simulations were ran on a Intel Core i7-9700 CPU @3.00Ghz, with 16GB of RAM.

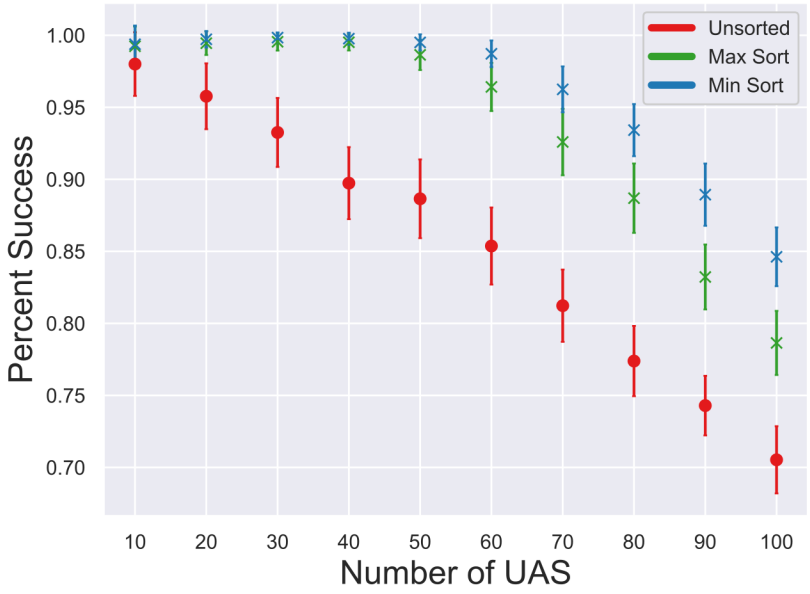


Figure 19: Success Rate and Standard Deviation

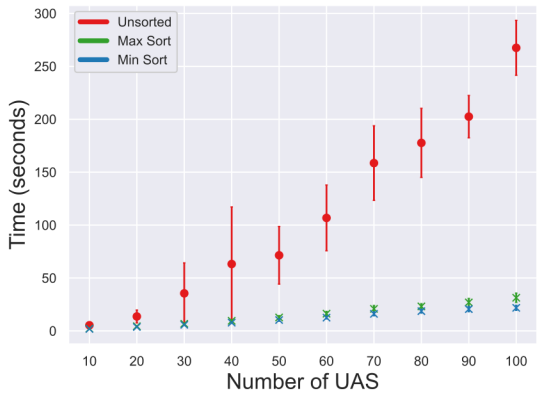
4.4.1 LoFi Simulation: There's no Music Here

For the Monte Carlo, the goal was to run numerous simulations to evaluate the performance characteristics of minimal, maximal, and unsorted prioritization methods (Figure 18) of the algorithm to compare success rate, time complexity, and space complexity. To do so, develop a map of size 100 x 100 x 75m was created with its predefined regions, corridors, and entrances. These corridors are set to be 5 units apart in the lateral (x,y) and vertical (z) directions of each other. Over 1000 simulations were then conducted ,from 10 to 100 UAS Operators, for *each* number of UAS, and for each of the three prioritization methods giving a total of 30,000 simulations. These UAS Operators have a uniformly distributed random starting point and goal point and are all requesting a path to be found at the same time instance. By having different numbers of UAS Operators it

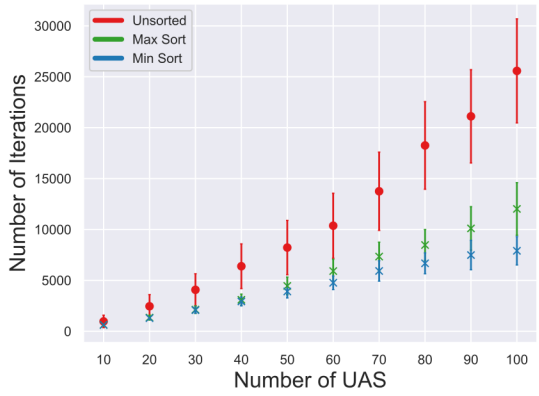
would allow us to understand the scalibility performance of the algorithm. In addition the start and end locations for the UAS were varied to simulate different UAS Operators that have their own intentions or operations to assess the algorithm. At the conclusion of each operation flight trajectories, iteration count, success rate, and time to find the path were all logged for evaluation.

After running all Monte Carlo simulations, the mean and standard deviations for each number of UAS were computed, depending on which prioritization was conducted. Figure 20 displays the mean and standard deviations of the success rate, time complexity of this compilation of results. It must be noted that Figure 20a and 20b has the time and number of iterations for both successful and unsuccessful paths planned for UAS. Figure 19 reveals that for operations with less than 50 UAS, the maximum and minimum prioritization success rate were 99%. With operations greater than 50 UAS, there is a difference in performance between the two. For 100 UAS the mean success rate for unsorted, maximum, and minimum prioritization were 71%, 78%, and 84% respectively. For time to find paths for all UAS, it was found that for 100 UAS the times were 275 seconds, 25 seconds, and 20 seconds for unsorted, maximum priority, and minimum priority respectively. Last for the total iterations, it was noted that for 100 UAS the total number of iterations to find a solution with unsorted, maximum prioritization, and minimum prioritization were 25,550, 11,920, and 7,710 total iterations.

Table 2 shows the mean percent quality of solution, derived from Equation 4.3, was found and plotted with the standard deviation for each method of prioritization with



(a) Time to Complete



(b) Number of Iterations

Figure 20: Time to Find Solution and Number of Iterations

the respective number of UAS conducted in the simulations. Optimal solution was considered as the straight line *lateral* distance or 100% optimal, this was then compared to the *lateral* solution length. This was done because the Weighted A* algorithm penalizes changes of height adjustments.

$$\%\ Quality = |\frac{actual}{optimal}| \tag{4.3}$$

Table 2: Quality of Solution Results

Number of	Unsorted		Max Sorted		Min Sorted	
UAS	Mean %	SD $\pm\%$	Mean %	SD $\pm\%$	Mean %	SD $\pm\%$
10	84.45	6.42	85.53	6.12	86.31	5.01
20	83.39	6.05	83.51	6.15	85.12	5.12
30	83.51	6.12	81.55	7.25	84.07	5.13
40	82.93	7.05	79.12	8.12	82.94	6.01
50	82.47	7.02	77.23	9.75	81.80	6.51
60	82.30	7.31	76.13	9.81	80.39	6.82
70	81.94	7.51	75.75	9.95	79.44	7.10
80	81.65	7.52	75.59	9.85	78.41	7.49
90	81.66	7.61	75.53	9.82	77.76	7.56
100	80.12	8.03	75.21	9.75	77.75	8.25

Results from Table 2 reveals that the unsorted method yields the best quality of solutions. At 100 UAS the unsorted method gives 80.12% optimal solutions in comparison to max and min which give 75.21% and 77.75% respectively. This reasoning could possibly due to the pure randomness of planning of the operations for the unsorted method. For the max and min prioritization, quality of solutions for both degrade as the number of UAS operations increases. However, min prioritization degrades in a slower manner than max prioritization.

These results indicate that minimum prioritization has the best performance metrics in comparison to utilizing max prioritization. The minimum prioritization gives a 6% increase of success rate for 100 UAS, a 25% performance increase for time to find solutions, and a 15% reduction of iterations. Both the time and number of iterations have

characteristics of $O(n)$ correlated with the number of UAS. In addition, failures happened when corridors within a region of the UAS requesting the path are all occupied, preventing the UAS from finding a abstract waypoints during the high level search. Furthermore it must emphasize that failures are when a solution path cannot be found for a UAS, and is not because of collisions between UAS. Regardless, of which method prioritization was utilized there is a linear degradation of success rate with the increase of UAS, which could be a relation towards the volume of airspace allocated as well as the available corridors. After evaluation of the performance metrics from the Monte Carlo the algorithm was then implemented into the High Fidelity simulation utilizing minimum prioritization, which is discussed in the next section.

4.4.2 Hifi Simulation

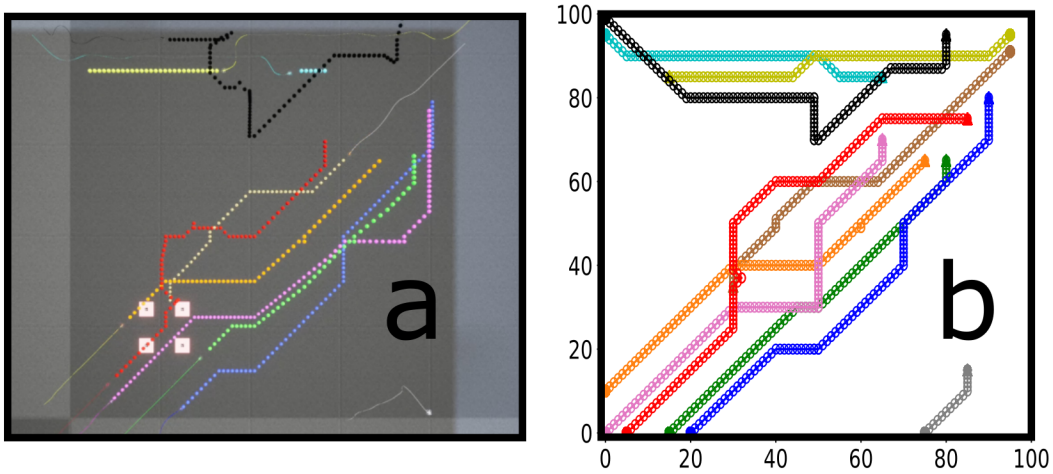


Figure 21: **Initial Multi-UAS Path Planning** **(a)** showcases the paths planned for the various UAS queries utilizing the algorithm and **(b)** is the 2-dimensional overlay of these paths planned for the UAS. It must be noted that these two images are not scaled respective to each other.

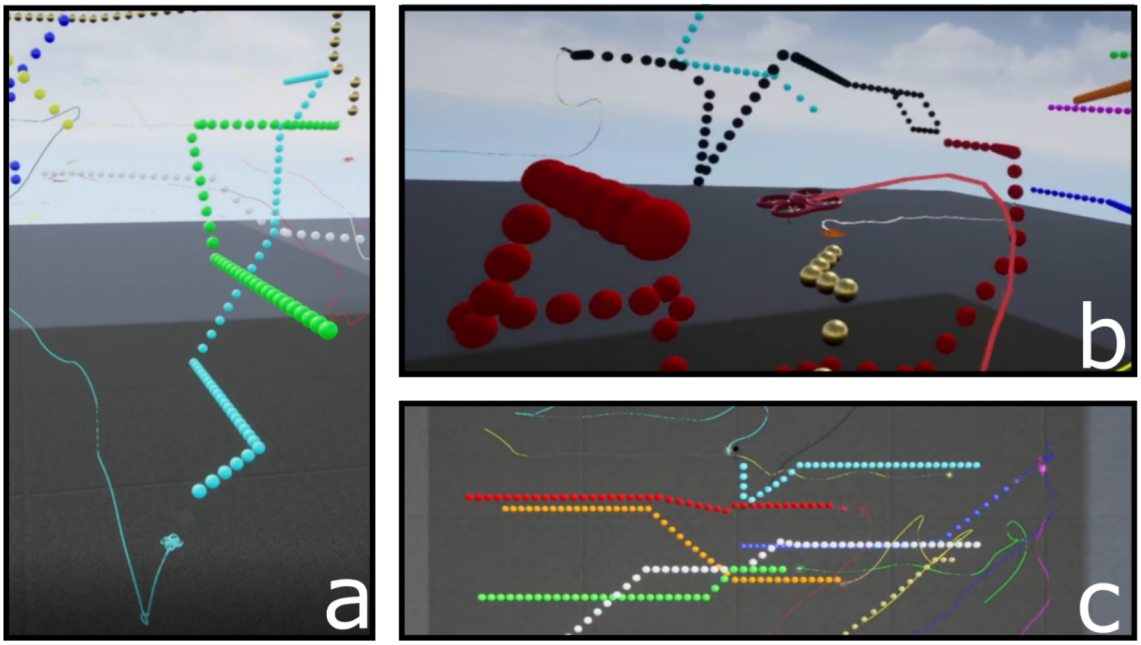


Figure 22: **UAS Traversal** **(a)** Cyan UAS traversing through the waypoints (visualized with cyan spheres) planned by the USS Path Planning Service, **(b)** Red UAS traversing through its respective waypoints (visualized with red spheres), notice how the waypoints have no intersections with each other, guaranteeing no collision from other UAS, **(c)** showcases the randomized paths queried by the UAS after the initial goal-point has arrived.

For the High Fidelity, the objectives were to evaluate the ease of integration with the notional UTM as well as have qualitative analysis of the performance of the algorithm with a simulated flight controller. The High Fidelity simulation framework discussed in Chapter 3 was utilized, and the architecture and framework is shown in Figure 23. As previously mentioned, the results from the Monte Carlo simulation were then leveraged and min prioritization was set into the USS Path Planning Service. Afterwards, a test scenario of 10 UAS requesting to have a path planned based on a given waypoint was ran.

The USS Path Planning Service would listen for incoming queries of UAS and would plan for these waypoints. After the goal was found, the UAS would query for 2 more random goal locations and the process would then repeat, all information sharing and exchange was done with MongoDB. If a path was not found, the UAS would be disapproved and ordered to standby until a solution was found. For the 2 random goal-point destinations it was specified that the lateral distances length to be greater than 50, or half the map. This is to simulate heavier density of autonomous air traffic, due to the hardware limitations. Visual results are displayed in Figures and 21 and 22 . During the simulations, operations were successful, with no collisions between UAS from occurring during traversal of UAS. Although the operations were successful, it must be emphasized that this path planning algorithm is for prior takeoff. In other words, this algorithm does not operate in a real-time environment for obstacle avoidance, and if a rogue UAS or random obstacle was placed in the space during operations, collision will happen. It is suggested incorporating the method with a real-time algorithm to reach these low level waypoints to guarantee safety of UAS traversal. In addition, velocity, and thus time is not implemented in this algorithm currently in the future introducing time to make a 4-d search space will be evaluated, to further evaluate this algorithm.

In addition, heavy CPU throttling occurred when attempting to simulate more than 15 UAS, hence why simulation tests were conducted with only 10 UAS.

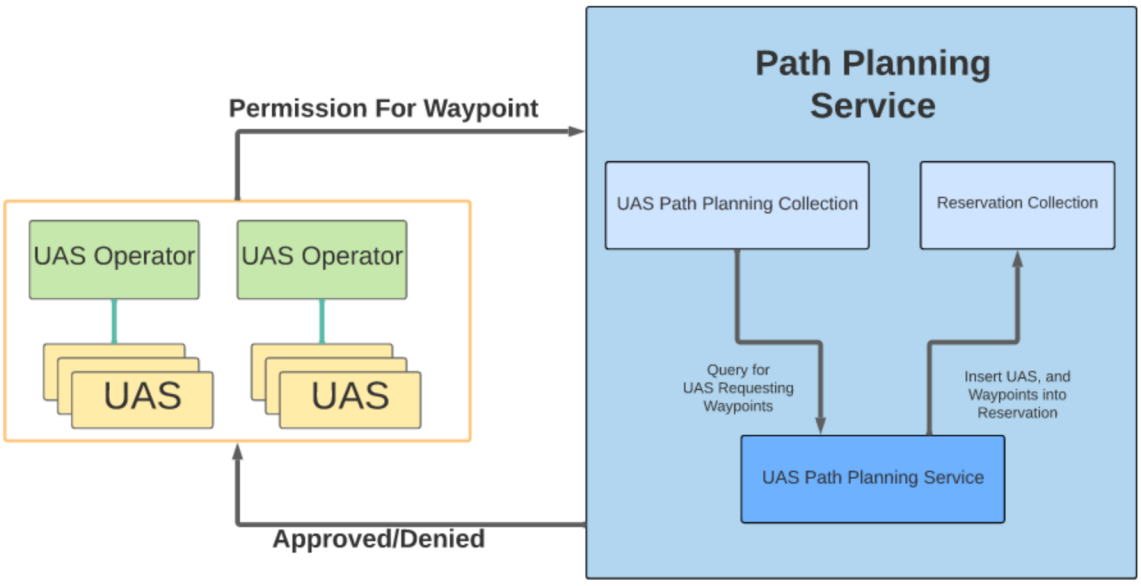


Figure 23: USS Path Planning Service

4.5 Conclusion

In this chapter, a multi-UAS path finding algorithm that implements the same procedures of path planning for manned aircraft protocols was proposed and implemented. This algorithm was tested utilizing both Monte Carlo and a High Fidelity simulation. From the Monte Carlo it was found that for operations under 50 UAS, success rate was 99%. Minimum prioritization for UAS generated the best results for path planning of operations. In the High Fidelity simulation the algorithm was incorporated into the notional UTM architecture developed by the FAA. The framework of this simulation was built utilizing Unreal Engine, Airsim, and ROS. These simulations were conducted with 10 UAS querying for waypoints to a goal destination. Results found no collisions during the operations of these simulations, however it must be noted that this method is for

prior takeoff, and to guarantee safety, coupling this method with real-time path planning algorithm would be the best course of action.

CHAPTER 5

LINEAR QUADRATIC GAUSSIAN (LQG) DESIGN FOR APRITAG TRACKING WITH A QUADCOPTER

5.1 Overview of this Chapter

This chapter discusses overall design process and implementation of a Linear Quadratic Gaussian (LQG) system which consists of a Kalman Filter and a Linear Quadratic (LQ) feed forward controller for tracking and landing of an AprilTag by a quadcopter. Motivation and procedures are outlined, with an in depth discussion of the design process of building the Kalman Filter and the LQ feed forward controller, as well as testing the system.

5.2 Motivation and Overall Procedures

Proper and safe landing protocols are fundamental for the operation of UAS, particularly for quadcopter systems. Most systems utilize GPS estimation to conduct landing, however this method can induce drift especially over a long continuous time during traversal. In addition, GPS denied environments prevent UAS from implementing this protocol, as well as areas where there are large buildings or structures, giving inaccurate readings of GPS location. To combat this, precision landing protocols are conducted utilizing fiducial tags, which are discussed in the 2.3.4. However these studies do not go in depth on the robustness of the tracking and landing guidance of the control law against disturbances,

as well as the design process formulation. From this lack of information the following is conducted for the development of the tracking and precision landing for a quadcopter system, so that in the future others can use this as a reference:

1. Develop a high level architecture for tracking and landing protocols for AprilTags.

2. Develop an optimal and robust control law for the guidance and control of the quadcopter to follow and land on the AprilTags.

3. Apply corrections to the AprilTag detection estimation, subject to error whenever the camera z and the AprilTag z axes are not aligned.

4. Test and compare the results of these corrections from video replays.

5. Design a Kalman Filter, with position of the AprilTag only known, and compare different Q models for visual tracking from video replays.

6. Design a LQ feedforward controller.

7. Compare $nominal$ and $visual$ tracking of the LQ feedforward control to a cascade control system, with disturbances incorporated in simulation tests.

5.3 Precision Tracking and Landing Architecture

Figure 24 displays the the architecture of the precision tracking and landing for the system, which is conducted through MAVROS protocol to communicate with the PX4 flight controller. For the tracking procedure, if the AprilTag is detected, reference tracking is conducted with a Kalman Filter as the observer and this reference point is sent to LQ

feedforward controller. During the tracking process, if the the target is lost, the Kalman Filter is utilized to feed in the last estimated reference point to the system for 5 seconds. After 5 seconds of tracking, the system will hover and await further commands. If the target is found again, then tracking continues.

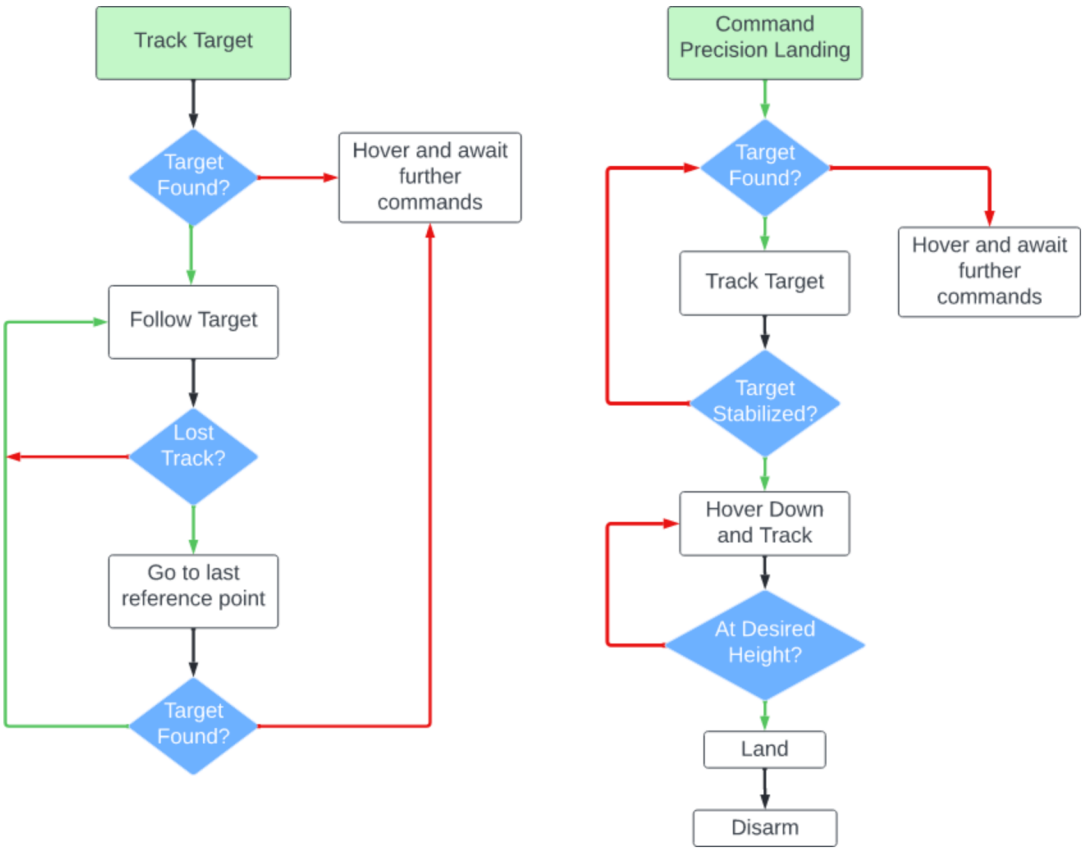


Figure 24: Visual Tracking and Precision Landing Architecture

For precision landing, the procedure utilizes the tracking target procedure to maintain precise landing of the quadcopter. If the quadcopter at any time loses detection of the AprilTag, then the system utilizes the Kalman Filter to track the last known location of the target for 3 seconds, while descending down at a constant rate speed of 0.5 m/s. If the

target is not found, then the quadcopter will fly up, hover, and await further commands from the user. If the target is found, the quadcopter will continue to descend down until at a desired height of 0.25m from the target, land, and finally disarm.

5.4 High Level Control

Figure 25, displays the structure of the control law, utilizing a Linear Quadratic (LQ) controller with feed forward and a Kalman Filter to observe the AprilTag. This particular structure is known as a Linear Quadratic Gaussian or LQG. This is because both systems attempt to minimize a quadratic function (consisting of Q's and R's) in a linear system, that has some Gaussian white noise. Because an optimal controller is required,

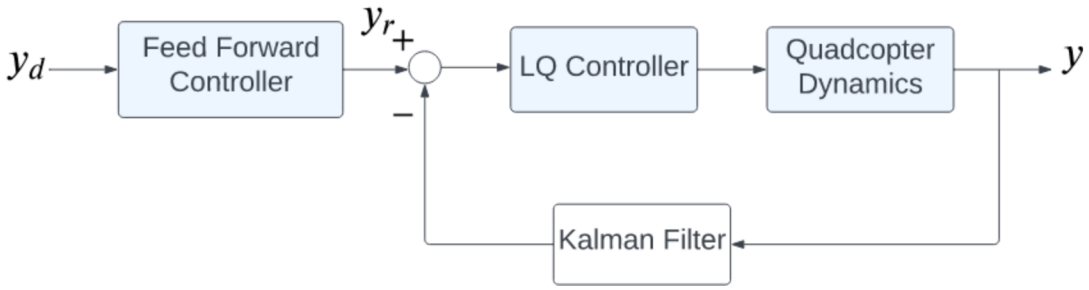


Figure 25: High Level Control

LQG, offers this, since it seeks to minimize some cost function for both the observer and the controller. In addition another requirement is robustness, that is the system provides stability despite uncertainties, disturbances, and noises injected in. Although robustness is not guaranteed [10], the LQ controller provides good robustness margins [3]. Last, LQG has a favorable property in that the control law and the observer can be designed

independent of each other, which is known as the separation principle. This allows the design process to be modular and allow decoupling of the systems for easier troubleshooting processes. In addition, to reinforce reasoning behind utilizing LQG, a study from [27], implemented an LQC controller for visual tracking of a target and controlling a robotic arm to command a reference position, with promising results. This gives an opportunity to design a novel LQG to track not the quadcopter, but to track a visual target as a reference input to the control for the quadcopter.

5.5 Visual Tracking Methodology

5.5.1 Camera Calibration

The overall goal of camera calibration is to receive all the information about the camera's intrinsic (internal), extrinsic (external) parameters, and the distortion caused by the lens of the camera. For the scope of this subsection, discussion will be on how to find the intrinsic and distortion parameters of the camera. In order to understand how this process works the standard pinhole camera model must be investigated [36]. Consider a camera system that consists of a barrier and film or sensor on the back. In the barrier there is a small hole (hence pinhole) that allows one or a few rays of light to pass through the barrier from the small hole. This allows mapping of the object in a one-to-one relation to the film or sensor. From this, an image plane can be found as shown in Figure 26. Now with this mapping, notation can be defined as follows:

- The **image plane** is defined as $\pi^{'}$.
- The **pinhole** or **center of the camera** is defined as O.

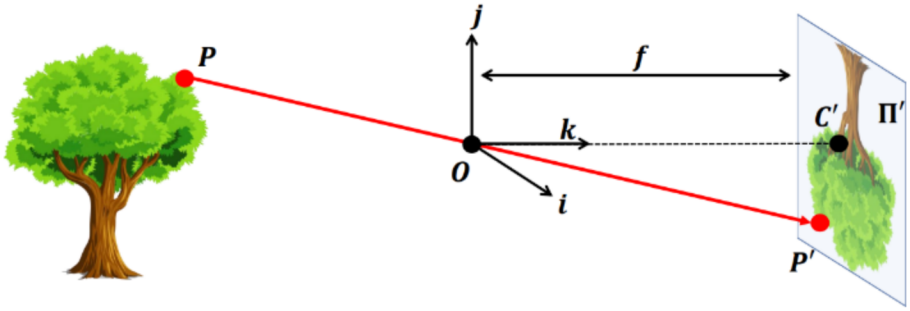


Figure 26: Pinhole Camera Model

- The distance between the pinhole, O and the image plane, is f.
- P is the point of the 3D object to the pinhole and can be defined as $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$.
- P will then be mapped to the image plane $\pi^{'}$, and in return: $P^{'} = \begin{bmatrix} x^{'} & y^{'} \end{bmatrix}^T$
- In addition O can be projected to the image plane $\pi^{'}$, as $C^{'}$, this line between $C^{'}$ and O can be notated as the **optical axis**.
- The camera **camera coordinate system** or the **camera reference system**, from the center of the pinhole O can be defined as $\begin{bmatrix} i & j & k \end{bmatrix}^T$.
- It can be noticed that there is a triangle formulated $P'C'O$ and P, O and the origin point of (0,0,z), with similar triangles and high school trigonometry $P^{'} = \begin{bmatrix} x^{'} & y^{'} \end{bmatrix}^T = \begin{bmatrix} f\frac{x}{z} & f\frac{y}{z} \end{bmatrix}^T$.

With the last statement, the equation for the intrinsic camera is as displayed in

equation 5.3, where c_x and c_y are x and y the center location of the camera:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{5.1}$$

With all these notations and derivations explained, the reality is that a pinhole camera is not practical for application since the pinhole is subject to the size of the hole: the bigger the hole more light rays can pass through which causes blurring of images. With a small hole size, the images become sharper, but become darker which detriments object detection processes. To combat this, lens, are utilized to address this issue. By applying lens to the system, this allows all rays of light to converge to P', which is shown in Figure 27, due to the lens refracting the rays.

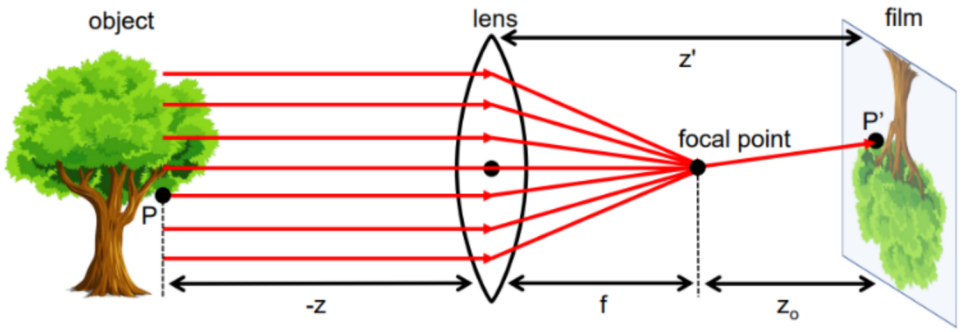


Figure 27: Lens Camera Model

However not all rays are refracted, if a point Q is beyond or too close to the lens' focus then blurring occurs. This limitation of the minimum and maximum ranges is known as depth of field. Furthermore, because of the concavity of the lens, this could cause distortions to the image. Figure 28, displays the barrel distortion and pincushion

distortion which are considered positive and negative distortions. To adjust for these

Figure 28: Lens Distortion

distortions a distortion matrix, K, is utilized:

$$K = \begin{bmatrix} k_1 & \dots & k_n \end{bmatrix}^T \tag{5.2}$$

With all the fundamental theory discussed, as recalled the whole idea is to find the intrinsic and distortions of the cameras. This was done using a planar checkerboard [43], where the camera rotates around the checkerboard and various different orientations and views. During this calibration, the the Harris-Corner detection algorithm is conducted to find the corners and edges of the checkerboard, lines are also extracted from the image plane. To apply this method with the simulation framework, a quadcopter was flown around with a camera attached. Calibration testing is shown in Figure 29.

During the application, the simulated quadcopter was flown around in various different angles from the checkerboard for the calibration process. Once this calibration

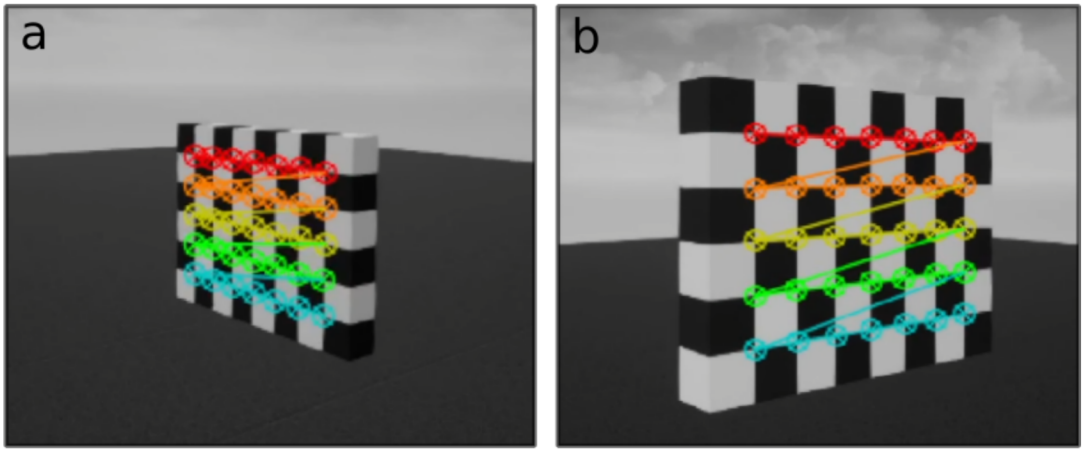


Figure 29: Calibrating the Simulation Camera

was conducted the calibration returns the values of Equation 5.3 and Equation 5.4.

$$\begin{bmatrix} 642.057 & 0 & 639.58 \\ 0 & 642.057 & 360.72 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

$$K = \begin{bmatrix} -0.002 & 0.002 & 0.0 & 0.0 \end{bmatrix}^T \tag{5.4}$$

These values are all derived from a 1280 x 720 pixel camera in the simulation. With the intrinsic and distortion matrix verified and calibrated, the extrinsic properties of the camera and its rotation relative to the quadcopter is discussed in the next section.

5.5.2 Rotation Matrices: Everything Has to be Relative

Figure 30, shows an image of the world, body(quadcopter), and camera frames in the environment. The overall goal is to have the relative position of what the camera image sees with respect to the body frame. To start off let the common rotation matrices around the z, x, and y axes be shown and derived from [19], where ψ, θ, and ϕ are yaw,

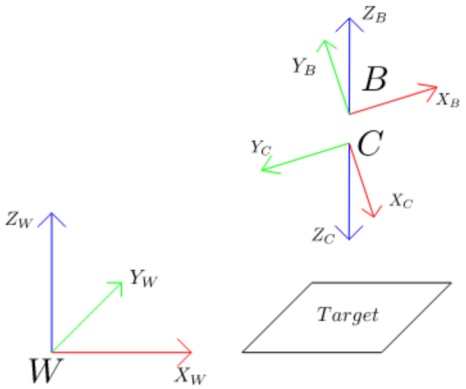


Figure 30: Coordinate frames between World, W, Body, B, and Camear, C

pitch, and roll.

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{5.6}$$

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{5.7}$$

Now let the the following statements be made:

- Let $R = R_z R_y R_x$.
- Let notations of rotation matrices from one frame , A, relative to another frame, B, be defined as R_B^A.

- Let the world, W, and the body or quadcopter, B be oriented in East North Up convention (ENU), this is done because ROS's world frame is defined in this convention, to switch to NED for the flight controller, MAVROS protocol converts the ENU frame of the B to NED.

- The camera frame C, is mounted on the bottom of the quadcopter, and is fixed.

- The position of the target or AprilTag is returned in reference of the camera frame C, because of this R_T^C is an identity matrix to the C.

The goal is to find R_T^B, that is the target position, T relative to the body frame, B: this can be done by the following:

$$R_T^B = R_T^C R_C^B \tag{5.8}$$

$$R_T^B = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} R \tag{5.9}$$

5.5.3 AprilTag Distortion Correction

Although AprilTags provide a robust framework to estimate pose and position, the estimation of the pose and orientation relies heavily on the orientation of the camera's z-axis in relation to the AprilTag's z-axis [2]. The study from [2] also notates that error is propagated from frame inconsistency from the motion. To fix the error from orientation a solution proposed is to apply a "Soft Yaw Axis Correction" technique, where the strategy is to apply trigonometric relations to apply a correction to the errors. To further mitigate errors, the study suggests to use a yaw axis gimbal with the camera, to keep the camera z

axis always aligned towards the center of the tag, as well as offer stabilization. The latter proposal could not be applied to this current system since the quadcopter implemented in the simulation, the only correction applied is utilizing a variation of the trigonometric relations proposed in the study for the application of tracking. To visualize the process

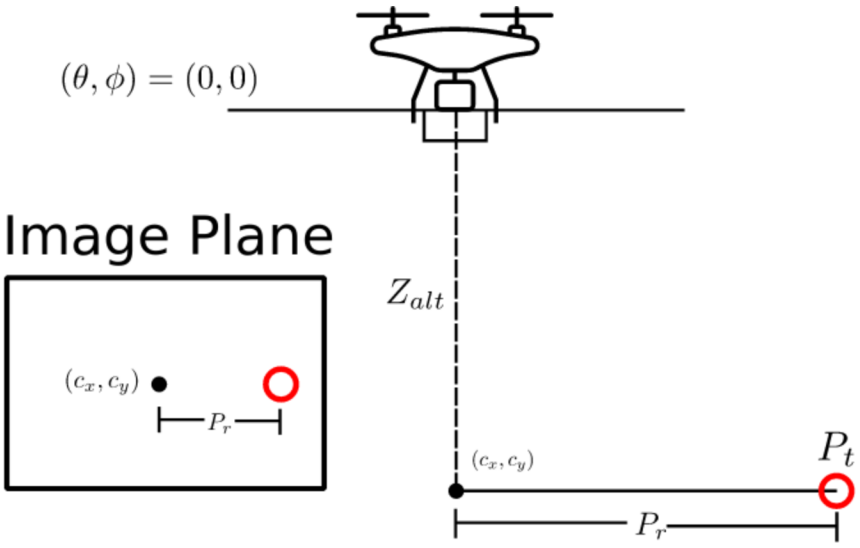


Figure 31: Level Flight and the Image Plane with the true read position of the target, P_t

Figure 31 is shown. To start off let the following statements be made:

- Z_{alt}, is the height of the quadcopter in reference to the ground, this information can be gathered from the barometer.
- P_t, is the ”true” relative position of the AprilTag, consisting of (p_{tx}, p_{ty}).
- P_r, is the measured relative position of the AprilTag, consisting of (p_{rx}, p_{ry}), read from the camera.
- The camera as recalled is *fixed* to the bottom of the quadcopter.

- The camera's center location (c_x, c_y) are shown.
- The camera's optical axis is *perpendicular* to the body frame attitudes (θ, ϕ).
- In Figure 31, the image plane showcases the seen image of the AprilTag and P_r, from (c_x, c_y) .

At level flight, when $(\theta, \phi) = (0, 0)$, P_r, can be be utilized as the main source for tracking. However during lateral flight, the quadcopter must pitch and or roll to reach to the desired position, this causes an attitude change, and causes misreadings of the AprilTag, shown in Figure 32. Where the quadcopter pitches at a certain angle θ_c. During attitude changes the AprilTag is assumed to be closer than it actually appears causing the tracking protocol to believe it is close to the target, but when the quadcopter levels itself, the values are incorrect. This causes a "wobble" effect for the tracking of the AprilTag, because of the inconsistent readings. To apply the the solution a new variable, the "distortion error" or δ, is introduced and consists of (δ_x, δ_y).

From Figure 32, P_t can be found from the following equations:

$$P_t = P_r + \delta \tag{5.10}$$

$$(p_{tx}, p_{ty}) = (p_{rx}, p_{ry}) + (\delta_x, \delta_y) \tag{5.11}$$

$$\delta_x = Z_{alt} \tan\theta \tag{5.12}$$

$$\delta_y = Z_{alt} \tan\phi \tag{5.13}$$

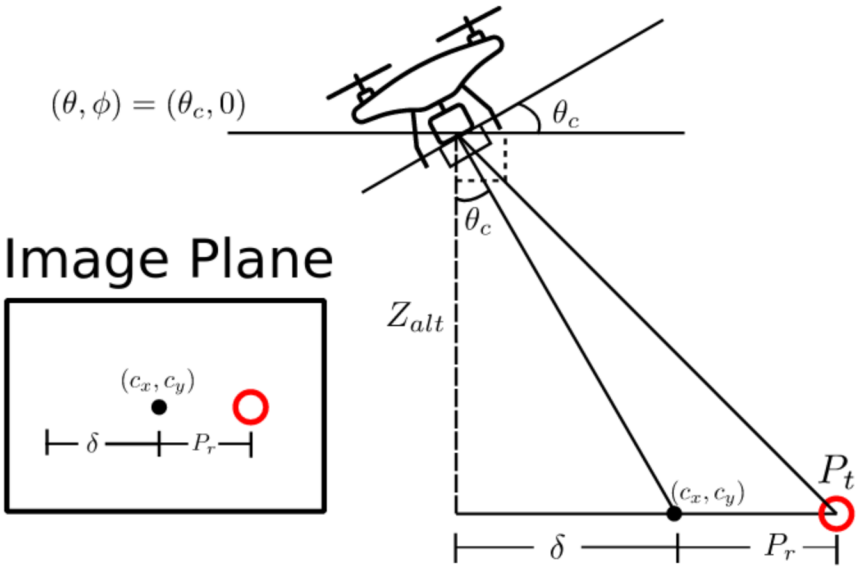


Figure 32: Attitude Change and the Image Plane with the True Read Position of the Target, P_t

5.5.4 Testing the Distortion Correction

To test the distortion correction, replays of three videos from ROS, were utilized. In each of these videos there consists three specific movements of a quadcopter tracking an AprilTag. The video replays are as follows:

1. A stationary track where the quadcopter identifies an AprilTag and begins traversal to it, shown in Figures 33a to c.

2. An attitude and slight position change from wind disturbances applied to quadcopter, with a AprilTag underneath the quadcopter shown in Figures 33d to f.

3. A forward tracking video where the quadcopter tracks a moving AprilTag, shown in Figures 33g to i.

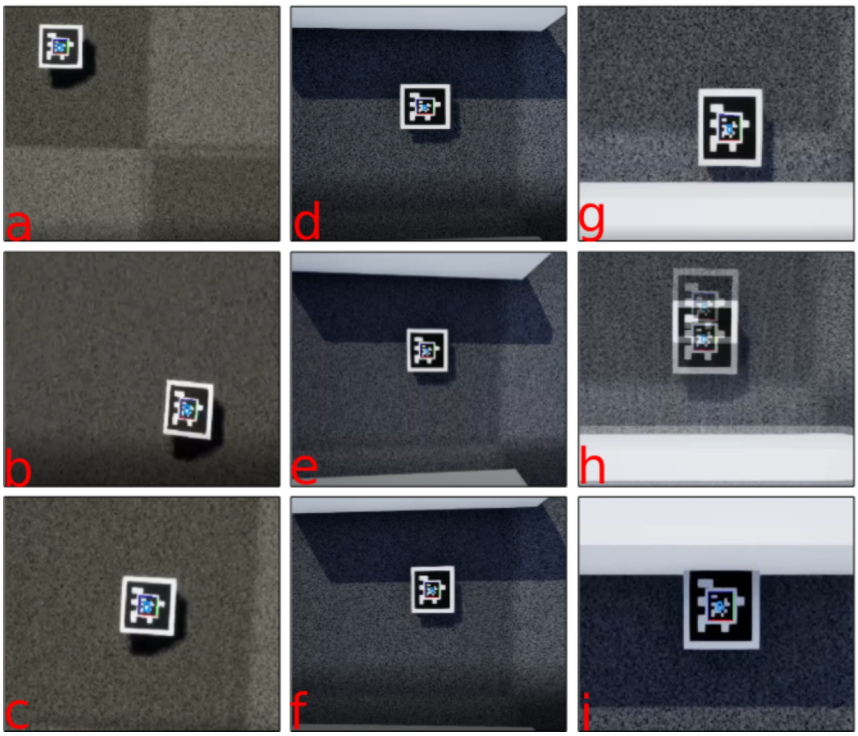
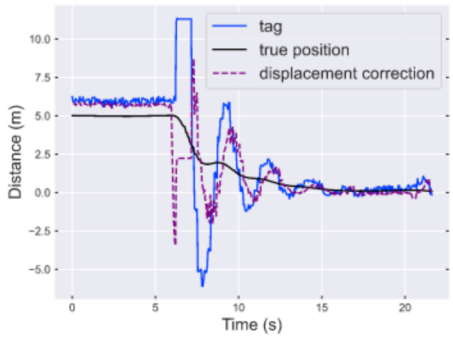
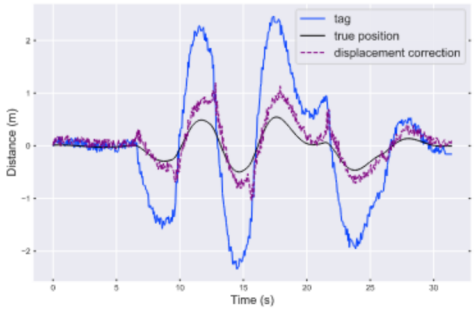


Figure 33: The Three Videos used for Testing

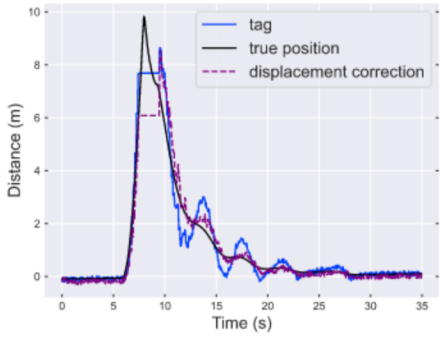
These replays from ROS, or ROSbags, were used because all information and topics related to the AprilTag, quadcopter position, and various other topics are saved. This allows the distortion script to subscribe to the raw AprilTag topic and apply the distortion correction and provide fair comparison between the two. During testing, the relative positions of the true relative, raw relative, and distortion relative positions were recorded and plotted for qualitative analysis. In addition, the Coefficient of Determination, R^2, and the Root Mean Squared Error, RMSE, were calculated for quantitative analysis.



(a) Distortion Correction and Raw for Station Tracking



(b) Distortion Correction and Raw from Attitude Changes



(c) Distortion Correction and Raw for Mobile Tracking

Figure 34: Tracking Stationary Target and Disturbance Tracking

5.5.5 Results of Distortion Correction: 1 or 2?

Results of the tests are plotted and shown in Figure 34. Visually it can be seen that during the tracking, the distortion correction technique follows the true relative position of the AprilTag in comparison to the raw sensor data for all tests. The raw information of the position is suspect to large errors, as seen in Figure 34b. As recalled during this test, the

quadcopter is subject to changes of attitudes, and thus it can be verified that the AprilTag errors do propagate from aggressive change of attitudes from the change of orientation of the fixed camera.

Table 3: R^2 and RMSE (m) of Distortion Correction (DC) and Raw Detection

Detection	Station Track		Attitude Track		Moving Track	
Method	R^2	RMSE	R^2	RMSE	R^2	RMSE
DC X	0.620	1.804	0.685	0.241	0.901	0.62
DC Y	0.704	1.443	0.731	0.222	0.303	0.083
Raw X	0.503	2.836	0.369	0.917	0.947	0.509
Raw Y	0.541	2.489	0.372	0.922	0.23	0.066

As recalled, the R^2 and RMSE values were recorded after testing was conducted, values for all three tests with the distortion correction and raw sensor detection were tabulated in Table 3. To understand R^2, it a statistical method to determine how close the model fits to the data. It can defined as the ratio explained variation over the total variation or in essence the sum of the square of the residuals over the total sum of the squares. This value of R^2 ranges between 0 and 1, where 1 is considered a perfect fit and 0 meaning poor fit. As a rule of thumb R^2 values greater than 0.6 are considered a "good" fit between the model and the data, or showing a high level of correlation. For RMSE, it is the summation of the square root of the errors between the predicted value and the actual values. The higher the value, the poorer the fit, and in for this application RMSE is in units of meters. By utilizing R^2 and RMSE, this provides a broader interpretation and comparison between the two methods of detecting the AprilTag on a quantitative basis.

Results from Table 3 reveals that for the station and attitude track tests, the distortion correction method has a higher R^2 value and lower RMSE value. For R^2 values in the attitude track the distortion correction has almost twice the R^2 value compared to the raw detection. For the RMSE, the distortion correction has an error 4 times smaller than the raw detection. For the moving track tests, the raw detection method has a better R^2 value and a lower RMSE value. But this does not matter much since the R^2 values for both of these methods are greater than 0.9 which means a terrific fit for both and the RMSE values between the two are 0.13m in difference during the tracking of a moving target.

From these results, it can be concluded that prior to applying the Kalman Filter, **the distortion correction method should be applied**. For all three tests, the distortion correction applied to the AprilTag provided a strong fit in R^2 values, as well as minimal errors in comparison to the raw detection. This is crucial to apply before sending the information to the Kalman Filter, to ensure that tracking is at optimal performance and to prevent any instability from poor tracking reference.

5.6 Kalman Filter Synthesis

The Kalman Filter as recalled from he Literature Review in Chapter 2, is a state estimation technique, this is done by initializing the following parameters:

1. Define the state-transition model, F_k.

2. Define the state vector of the process time x_k.

3. Define the observation model, H_k.

4. Define the covariance of the observation noise, R_k.

5. Define the covariance of the process noise, Q_k

6. Define the control-input model, B_k, which can be added into the system, if B_k is added onto the system then define u_k, the control vector as well. For this application there is no B_k since there is no knowledge of the controller of the target.

From this the following equations can then be derived:

$$x_k = F_k x_{k-1} + B_k u_k + w_k \tag{5.14}$$

$$z_k = H_k x_k + v_k \tag{5.15}$$

Where w_k in Equation 5.14, is considered the process noise of some normal distribution, with a covariance of Q_k. In Equation 5.15, v_k is the observation noise, which has characteristics of a Gaussian white noise, from the covariance of R_k. From this the main two procedures of the Kalman Filter, the **prediction** and **update** phases, can now be explained. The prediction step has two intermediate steps:

1. Predict the state estimate:

$$x_k = F_k x_{k-1} + B_k u_k + w_k \tag{5.16}$$

2. Predict the error covariance:

$$P_k = F_k P_{k-1} F_k^T + Q_k \tag{5.17}$$

For the update phase, it has 5 intermediate steps:

1. Compute the residual, y_k:

$$y_k = z_k - H_k x_k \tag{5.18}$$

2. Compute the Innovation Covariance, S:

$$S_k = H_k P_k H_k^T + R_k \tag{5.19}$$

3. Compute the Kalman Gain, K:

$$K_k = P_k H_k^T S^{-} 1_K \tag{5.20}$$

4. Provide corrections to the state estimates, x_k:

$$x_k = x_k + K_k y_k \tag{5.21}$$

5. Provide corrections to the state estimates, x_k:

$$P_k = (I - K_k H_k) P_{k-1} \tag{5.22}$$

These equations were coded and applied to take in the incoming position read from the distortion correction measurements of the AprilTag. In the next subsections, discussion will be on defining the parameters for the input, afterwards the design selection of the values and models will be rationalized.

5.6.1 Two Models: State Transition and H_K

For the state transition modeling, F, instead of utilizing a constant velocity, a constant acceleration model is applied, this would allow tracking of for changes of velocities for the target. Because of this, F becomes a (6 x 6) matrix for the position, velocities, and accelerations of the target.

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & dt & 0 & \frac{1}{2dt^2} & 0 \\ 0 & 1 & 0 & dt & 0 & \frac{1}{2dt^2} \\ 0 & 0 & 1 & 0 & dt & 0 \\ 0 & 0 & 0 & 1 & 0 & dt \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.23}$$

For the state vectors, x it is defined as follows:

$$x = \begin{bmatrix} x & y & \dot{x} & \dot{y} & \ddot{x} & \ddot{y} \end{bmatrix}^T \tag{5.24}$$

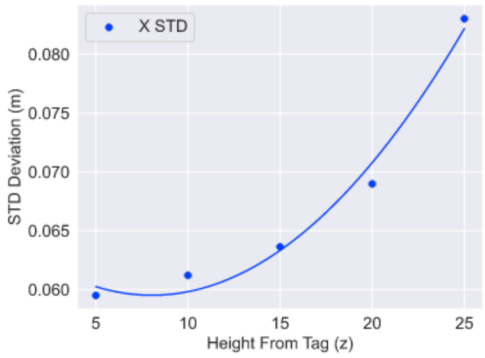
The observation model, H, a (2x6) matrix can be defined. where the 1's represent the measurement of the x and y location of the target.

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.25}$$

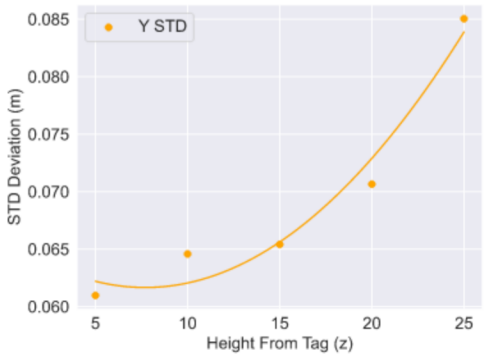
5.6.2 Sensor Noise Covariance R

For R, is a (2 x 2) size matrix since the only read values from the AprilTag is the position in the x and position in the y, and shown in 5.26. Where R_1 and R_2 are the covariance values for the x and y position from the sensor.

$$R = \begin{bmatrix} R_1 & \\ & R_2 \end{bmatrix} \tag{5.26}$$



(a) σ of AprilTag X, R^2 = 0.982



(b) σ of AprilTag Y, R^2 = 0.959

Figure 35: Results of Regression Fit of σ

As recalled, this is the *variance* of the sensor noise, and because the sensor is a camera tracking the position of an AprilTag, this noise will vary between height and orientation. Because it is known that the accuracy of the AprilTag is dependent on the depth of view from the camera to the fiducial tag, data was logged of the position estimates of the tag from the camera at various different heights. From this data, the standard deviations, σ, of the x and y: σ_x, σ_y were logged and plotted at different heights. Then a polynomial regression of the 2nd order was fit onto the σ_x and σ_y values and is shown in Figure 35. From this an R^2 was computed to determine how well the regression fit to the data. Results indicate that the R^2 for the regression was 0.982 and 0.959, indicating a strong fit to the data provided. From this the the equations for the 2nd order regression fit were utilized and put into the R matrix for R_1 and R_2 respectively.

$$R_1 = ((7.81E-05)z^2 - (1.26E-03)z + 0.0646)^2 \tag{5.27}$$

$$R_2 = ((7.42E-05)z^2 - (1.14E-03)z + 0.066)^2 \tag{5.28}$$

where z represents the height from the AprilTag, which will be obtained from the barometer of the quadcopter, and assuming that the AprilTag is "close" to the ground. The squared factor, is done because squaring σ gets the variance. With this, R is updated constantly based on any changes of height.

5.6.3 Process Noise Covariance Q

For Q, two matrices will be defined: Q_1 and Q_2. This is done to evaluate which Q model yields the best results of the observing the position of the AprilTag. Q_1 consists of a diagonal matrix where the values of each state are independent of each other.

$$Q_1 = \begin{bmatrix} Q_a & 0 & 0 & 0 & 0 & 0 \\ 0 & Q_b & 0 & 0 & 0 & 0 \\ 0 & 0 & Q_c & 0 & 0 & 0 \\ 0 & 0 & 0 & Q_d & 0 & 0 \\ 0 & 0 & 0 & 0 & Q_e & 0 \\ 0 & 0 & 0 & 0 & 0 & Q_f \end{bmatrix} \tag{5.29}$$

For Q_2, another model that accounts for the variance in acceleration, σ_a^2 can be described as follows:

$$Q_2 = \begin{bmatrix} \frac{\delta t^4}{4} & \frac{\delta t^3}{3} & \frac{\delta t^2}{2} & 0 & 0 & 0 \\ \frac{\delta t^2}{2} & \delta t^2 & \delta t & 0 & 0 & 0 \\ \frac{\delta t^2}{2} & \delta t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\delta t^4}{4} & \frac{\delta t^3}{3} & \frac{\delta t^2}{2} \\ 0 & 0 & 0 & \frac{\delta t^2}{2} & \delta t^2 & \delta t \\ 0 & 0 & 0 & \frac{\delta t^2}{2} & \delta t & 1 \end{bmatrix} \sigma_a^2 \tag{5.30}$$

These two models will be compared against one another and discussed in the next sub-section with the specified values.

5.6.4 Design Iterations of Kalman Filter and Results

For selection of the Kalman Filter, a test was conducted, similar to the methods discussed in section 5.5.4. For Q_1, values of 1E-1 to 1E-7 were set. For Q_1, these values indicate how much the model F should be trusted, the higher the value the less it should be trusted, due to a higher variance. From these specified values, the three videos from Figure 33 were used again, with the Kalman Filter from Q_1 logged. For Q_2, a range of values for σ_a^2 were set from 0.1 to 0.8 m^2/s with the process repeated as previously mentioned. The R^2 and RMSE values were all then computed and tabulated in Tables 4 and 5 for Q_1 and Q_2 respectively.

Table 4: Q_1 results with R^2 and RMSE, highlighted values are the best fits for each test

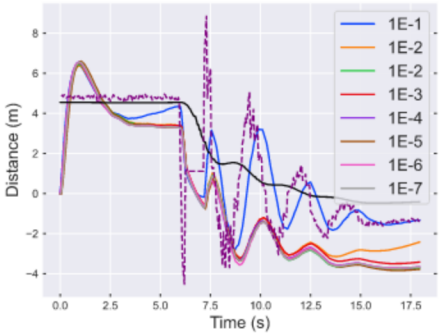
Q	Station Track				Attitude Track				Moving Track			
Q_1	R2		RMSE (m)		R2		RMSE (m)		R2		RMSE (m)	
σ	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
1E-1	0.633	0.699	1.6	1.329	0.53	0.57	0.382	0.359	0.947	0.689	0.401	0.062
1E-2	0.444	0.544	2.368	2.082	0.13	0.13	0.424	0.394	0.98	0.704	0.238	0.063
1E-3	0.379	0.477	2.702	2.425	0.27	0.74	0.384	0.366	0.984	0.71	0.21	0.066
1E-4	0.405	0.499	2.547	2.279	0.18	0.63	0.386	0.365	0.99	0.626	0.165	0.067
1E-5	0.369	0.461	2.69	2.435	0.21	0.72	0.39	0.367	0.991	0.517	0.158	0.07
1E-6	0.361	0.448	2.736	2.498	0.21	0.65	0.382	0.364	0.991	0.467	0.152	0.072
1E-7	0.368	0.463	2.705	2.442	0.16	0.63	0.388	0.363	0.987	0.458	0.189	0.075
1E-8	0.363	0.46	2.714	2.442	0.2	0.69	0.396	0.37	0.992	0.45	0.149	0.076

Table 5: Q_2 results with R^2 and RMSE, highlighted values are the best fits for each test

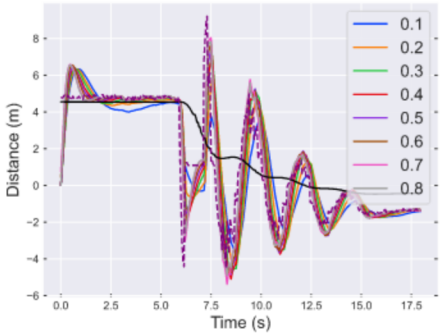
Q	Station Track				Attitude Track				Moving Track			
Q_2	R2		RMSE (m)		R2		RMSE (m)		R2		RMSE (m)	
σ	X	Y	X	Y	X	Y	X	Y	X	Y	X	Y
0.1	0.608	0.658	1.772	1.537	0.594	0.644	0.319	0.309	0.967	0.389	0.341	0.064
0.2	0.544	0.595	2.05	1.768	0.642	0.7	0.282	0.257	0.978	0.352	0.272	0.066
0.3	0.504	0.541	2.234	1.957	0.659	0.719	0.267	0.239	0.98	0.308	0.256	0.067
0.4	0.505	0.531	2.249	2.003	0.666	0.735	0.26	0.227	0.982	0.307	0.239	0.068
0.5	0.515	0.531	2.218	1.999	0.672	0.73	0.255	0.227	0.984	0.307	0.23	0.069
0.6	0.516	0.521	2.227	2.044	0.67	0.735	0.252	0.223	0.985	0.31	0.223	0.069
0.7	0.511	0.513	2.253	2.088	0.681	0.735	0.247	0.223	0.983	0.306	0.231	0.071
0.8	0.525	0.517	2.19	2.063	0.678	0.734	0.249	0.222	0.987	0.33	0.204	0.07

For the station track test, it can be seen that the Q_1 has the best overall R^2 and RMSE value in comparison to Q_2, when σ is set to 1E-1 across the diagonals of the Q_1 matrix. However it can be noticed that in Q_1, σ values from 1E-2 to 1E-8 have a significant drop in performance. For the Q_2 model, there is also a performance drop from 0.1 to 0.8 m^2/s, however it does not drop as significantly compared to Q_1. For the attitude track, Q_2 dominates in comparison to Q_1. Last for the moving track test, both Q_1 and Q_2 have good results for predicting the target moving, with the Q_1 model slightly outperforming the Q_2 model.

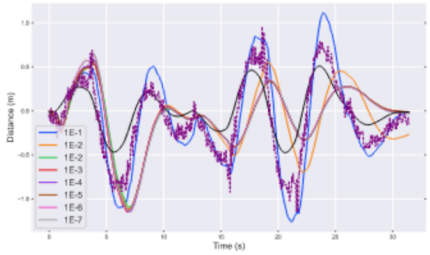
Although quantitative analysis reveals that the two have relatively close performances, a visual analysis is also conducted in Figure 37. Figures 37a and 37b reveals that Q_1 does a poor job tracking the position of the stationary AprilTag, and does not even follow the trend of the distortion correction in comparison to Q_2. Figures 37c and 37d also



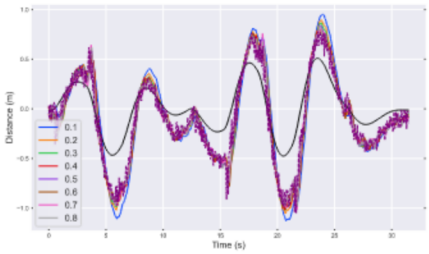
(a) Q_1 Position Station Track



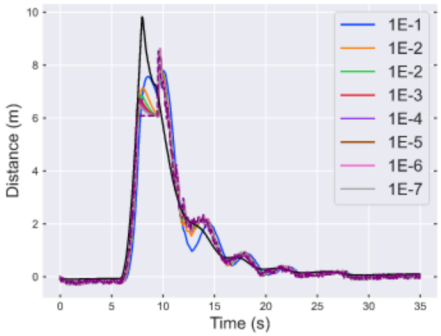
(b) Q_2 Position Station Track



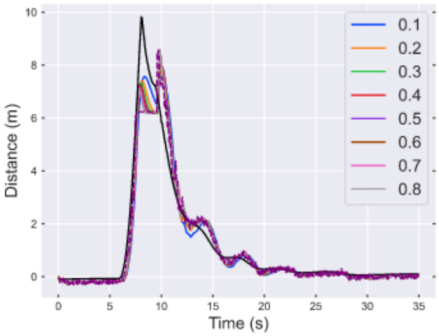
(c) Q_1 Attitude Tracking



(d) Q_2 Attitude Tracking



(e) Q_1 Moving Tracking



(f) Q_2 Moving Tracking

Figure 37: Visual Comparison between Q_1 and Q_2

displays a lag response in the model of Q_1, with lag increasing as σ decreasing. However Q_2 is able to follow the trend of true position of the AprilTag with different σ_a values. Finally, Figures 37e and 37f showcase that the two have relatively good performance for trracking the moving target. From the visual results, it can be concluded that the optimal Q model is Q_2, for its basis on being able to track the position of the AprilTag, in all three cases in a consistent manner in comparison to Q_1.

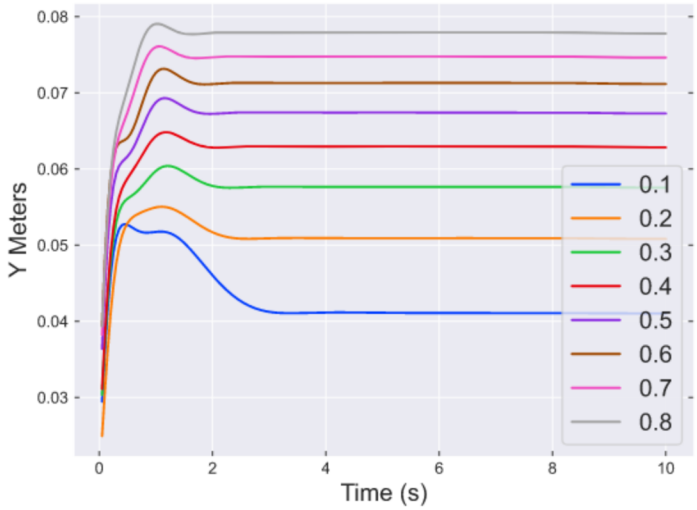


Figure 36: Predition Error Covariance, P, for the Y Position during Attitude Test

This leads to the question which σ_a value is to be chosen from Q_2. Results from Table 5, tell that the strongest performances from the three tests are values of 0.1, 0.7, and 0.8 m^2/s. One last evaluation will be done with the comparison of the P, the prediction error covariance. An optimal response is where P converges to the smallest value as time approaches to infinity. A poor response of P is if it never converges to a value and has inconsistent behaviors as time approaches infinity. Figure 36, is a plot of the P for the y

position during the attitude test, it must be noted that all other plots for the P values for x and y with different tests all had same characteristics as Figure 36. It can be seen that with a σ_a value of 0.1 m^2/s has the steepest descent of convergence in comparison to the other values. This in essence means that the Kalman Filter for this specific value has decreased to a small value to the point where the estimate will no longer change. Although Figure 36, shows that value a σ_a value 0.1 m^2/s has the lowest convergence, the values of the convergence for the different values are all relatively close to each other in spectrum. From the analysis of the plots, the tables, and the P performance and because value is placed heavily on tracking at different attitudes, 0.7 m^2/s was chosen for the value of σ_a.

With the distortion correct applied, and the Kalman Filter design finalized with the Q_2 model at a σ_a value of 0.7 m^2/s the control law synthesis can now be conducted.

5.7 Control Law Synthesis

5.7.1 PX4 Multicopter Cascaded Controller

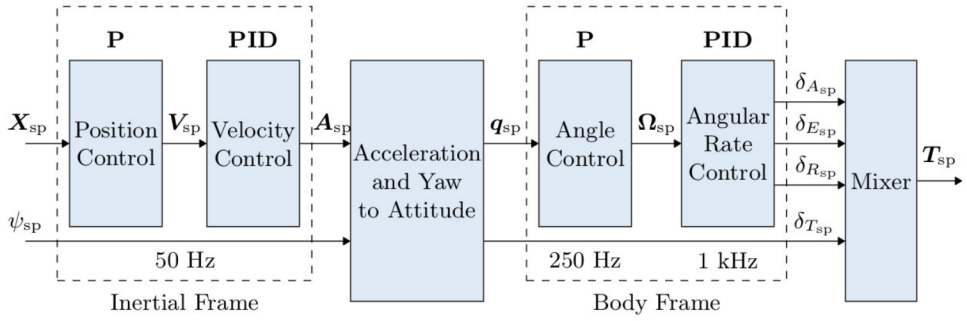


Figure 38: PX4 Cascaded Controller

PX4, is an open source autopilot system tailored towards research development of

control and guidance laws for autonomous systems of different configurations, as well as hobbyists. PX4 has a standard PID cascaded control law; shown in Figure 38, which was utilized to compare the performance of the LQ feedforward law developed. In this 2 part cascaded system, a reference position and or yaw position is set to a position controller with a P gain. This gain is set as the body velocity reference command and is sent into a PID velocity controller. The output of this velocity controller is then sent into an angular rate controller. The process is then repeated with a set angle fed into the angle controller and then a rate is fed into the angular rate controller, where the motor mixing commands are then fed into the system.

5.7.2 LQ Feed Forward

For the control law of the precision landing and tracking of the fiducial target, an LQ feed forward controller was designed and to be compared against a standard PID cascaded controller and its performances. To start off let a state space system be defined as:

$$\dot{x} = Ax + Bu \tag{5.31}$$

Let error of a state, x_e be defined as:

$$x_e = x - x_d \tag{5.32}$$

Where x, is the current state and x_d is the desired state. Let error of a controller input, u_e be defined as:

$$u_e = u - u_r \tag{5.33}$$

Where u, is the current input control and u_r is a reference input control. In order for trajectory tracking to be conducted, the controller, u_e must drive x_e to 0 with a compensator K. From this inputting equation 5.32 into equation 5.35.

$$u_e = K(x_e) \tag{5.34}$$

$$u_e = K(x - x_d) \tag{5.35}$$

Rearranging equation 5.33 and putting it into equation 5.35

$$u - u_r = K(x - x_d) \tag{5.36}$$

$$u = K(x - x_d) + u_r \tag{5.37}$$

With equation 5.37, the finalized control law can be defined as follows from equation 5.31:

$$\dot{x} = Ax + B(K(x - x_d) + u_r) \tag{5.38}$$

Figure 39 displays the block diagram schematic of the linear quadratic feed forward control law.

With this LQ feedforward control law defined, procedure of the implementation is as follows:

1. Define state space representation of the plant model, for this case a quadcopter.
2. Define Q and R values.
3. Solve the Algebraic Ricatti Equation to get S, to minimize the cost function, J.

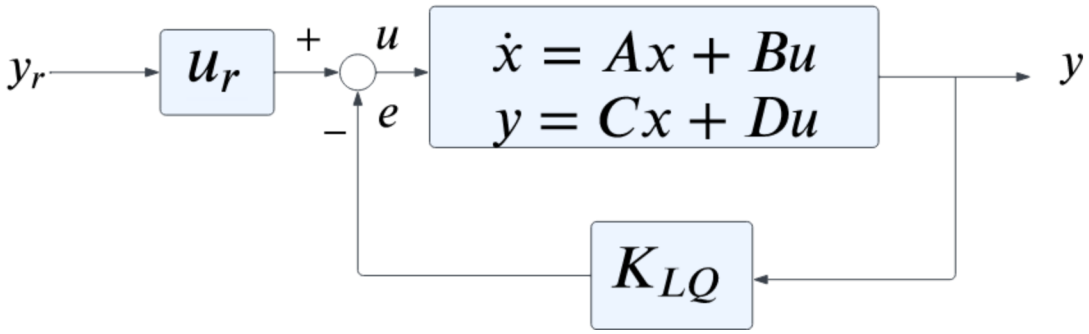


Figure 39: LQ Feed Forward Block Diagram Diagram

4. From S get compensator gains, K, from $K = R^{-}1B^{T}S$.

5. From the multiple K values, choose K solution that yields to a stable system, by looking at the eigenvalues that are on the left half of the pole plane.

Each of these procedures will be discussed in the next subsections of this chapter.

5.7.3 State Space Modeling of Quadcopter

The state space representation of a quadcopter has been heavily researched and conducted. Because LQ methods require a linear representation of the system and model, derivations of the quadcopter model from [40] and [37] were utilized to build and implement the LQ feed forward control in the longitudinal and lateral directions of the system. To control the longitudinal and lateral directions of the quadcopter only 8 state vectors from the 12 state model are needed as show below.

$$X = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} & y & \dot{y} & \phi & \dot{\phi} \end{bmatrix}^{T} \tag{5.39}$$

Because only 8 states are needed, the state space matrix **A** becomes an (8x8) and **B** is an (8x2) matrix. In **A**, g is the constant of gravity. In **B**, I_x and I_y are the moments

of inertia for the x and y axes respectively.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.40}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{I_x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{I_y} \end{bmatrix}^T \tag{5.41}$$

For the input, u, it is a 2 vector row, since only control of lateral and longitudinal positions are desired, which are pitch, θ, and roll, ϕ, inputs to the quadcopter.

$$u = \begin{bmatrix} u_\theta & u_\phi \end{bmatrix}^T \tag{5.42}$$

For u_r, the reference command will be set as the previous input u to system during operation.

$$u_r = \begin{bmatrix} u_{\theta r} & u_{\phi r} \end{bmatrix}^T \tag{5.43}$$

5.7.4 You're A JQRK

In essence, what makes LQ powerful is that it allows the user to define weights on the importance of reaching a particular state with the Q matrix and apply penalties to the amount of throttle or power inputted into the system through the R. This is done by minimizing the quadratic cost function J shown in equation 5.44.

$$J = \int_0^\infty (x^T Q x + u^T R u) dt \tag{5.44}$$

Q in this application is a (8 x 8) matrix, where the values in the diagonal of this matrix represent the weight of importance for the system to reach this desired state. For instance if q_1 had a value of 100 and q_3 had a value of 1, that would mean the user has defined that reaching q_1's state, the x position for a quadcopter, is 100 times more important than q_3, the pitch angle for a quadcopter.

$$Q = \begin{bmatrix} q_1 & & \\ & \ddots & \\ & & q_8 \end{bmatrix} \tag{5.45}$$

$$R = \begin{bmatrix} r_1 & \\ & r_2 \end{bmatrix} \tag{5.46}$$

R in this application is a (2 x 2) matrix since there are 2 control inputs from u. R, as previously discussed penalizes the controller on its actuation specified by the value defined. The higher the value the heavier the penalty and the slower the control input response. For the application of the quadcopter system, Q and R are diagonal values that are to be defined. From these set values, the compensator gains, K, an (8 x 8) matrix can be found by the following equation:

$$K = R^{-}1B^{T}S \tag{5.47}$$

Where S is the solution matrix of size (8 x 8) found from the J utilizing the Algebraic Ricatti Equation:

$$A^{T}S + SA - SBR^{-}1B^{T}S + Q = 0 \tag{5.48}$$

The eigenvalues can then be found from using *MATLAB*'s or *Python*'s *eig* command to find which solution yields a stable system response. In Equation 5.49 K_n is a row vector

of size 8 and λ_n are the eigenvalue poles of the system for the n row of S.

$$\lambda_n = eig(Ax - BK_n) \tag{5.49}$$

The nominal values for the diagonal values of Q and R were found utilizing the Travis Duane Fields (TDF) Methodology, that is tuning Q and R from trial and error. 2 sets of Q and R values were set for nominal position tracking and vision tracking. For nominal position tracking Q_n and R_n:

$$Q_n = \begin{bmatrix} 9.1 & 0.1 & 0.1 & 0.1 & 9.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}^T \tag{5.50}$$

$$R_n = \begin{bmatrix} 14 & 14 \end{bmatrix}^T \tag{5.51}$$

For vision position tracking Q_v and R_v are set as follows, a higher penalty cost was set to the system due to instability of the tracking of the fiducial tag during the high-speed visual tracking:

$$Q_v = \begin{bmatrix} 1.8 & 0.1 & 0.1 & 0.1 & 1.8 & 0.1 & 0.1 & 0.1 \end{bmatrix}^T \tag{5.52}$$

$$R_v = \begin{bmatrix} 20 & 20 \end{bmatrix}^T \tag{5.53}$$

From these specified Q and R matrices, the method of finding the K gains, were found and tabulated below in Table 6, and formatted with the respective states, of the system.

5.8 Simulation Testing Environment

The same simulation environment from Chapter 3, utilizing Unreal Engine and ROS, was leveraged. PX4's Software-In-The-Loop (SITL) was utilized in the simulation

Table 6: K Gains for each State Vector

X	K_n	K_v
x	0.806	0.3
$\dot{x}$	0.385	0.185
θ	0.897	0.554
$\dot{\theta}$	0.123	0.097
y	-0.806	-0.3
$\dot{y}$	-0.385	-0.185
ϕ	0.897	0.554
$\dot{\phi}$	0.123	0.097

environment to control the UAS during flight. 5 tests were conducted to compare the LQ feedforward control against the PX4's standard cascaded control law:

1. A nominal position test, where the UAS traverses from origin (0,0) to (5,5), simulating a step input of 5.

2. Vision tracking with a *stationary* target at (5,5) and the UAS at the origin (0,0), simulating a step input.

3. Wind disturbance with a head gust wind of 8.94 m/s input to the simulation where the quadcopter must maintain stability and tracking of the target.

4. Vision tracking of a *moving* target, with the target starting at (-2,0) and moves to (8.35,0), simulating a ramp input.

5.9 Results and Discussion

5.9.1 Nominal Results

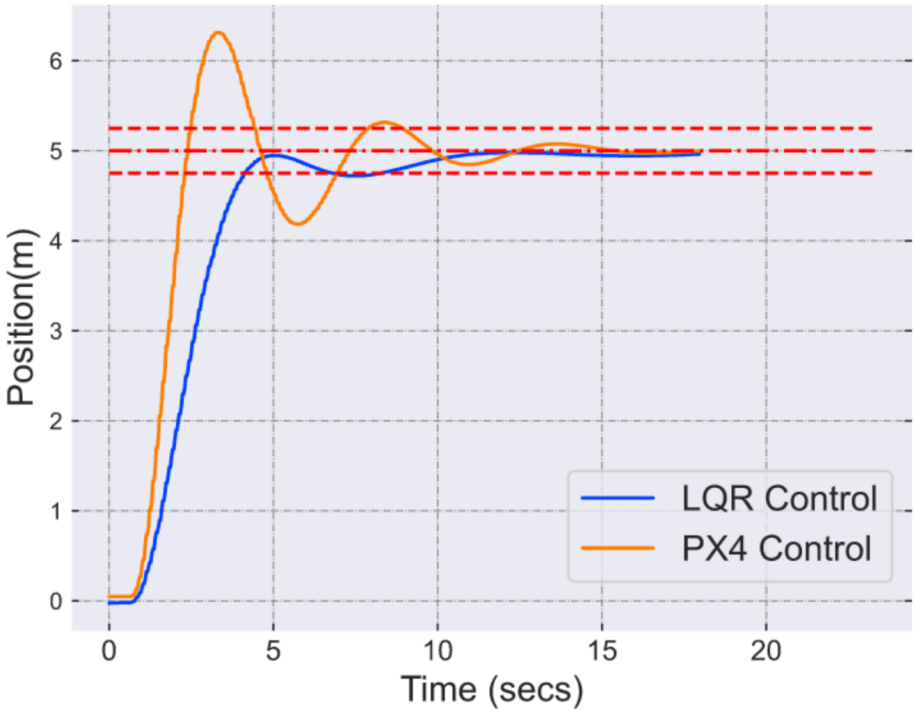


Figure 40: Nominal Position Tracking

The step input response in the x-directions for both the LQ and cascade control are shown in Figure 40. The y-direction results have the same response to the system due to the symmetrical dynamics of the quadcopter, as well as the weights and penalties of the Q and R matrices being symmetrical as well. Results from the step input reveals that the cascaded control has a percent overshoot of 22.5%, while the LQ system has no overshoot. However, in terms of rise time, the cascade controller takes 1.25 seconds, while the LQ controller, takes 2.75 seconds to reach the 90% value of 4.5m. Finally the

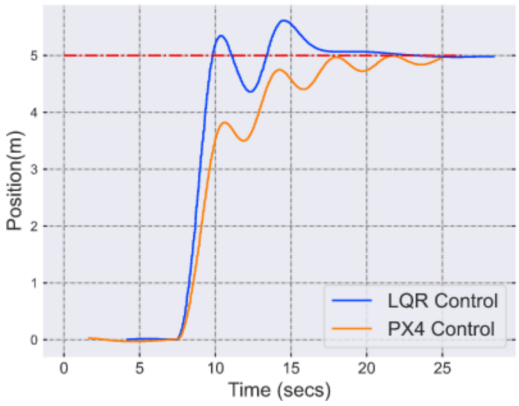
cascade control's settling time was 9.85 seconds, while the LQ settling time was 9.42 seconds.

5.9.2 Vision Tracking Results

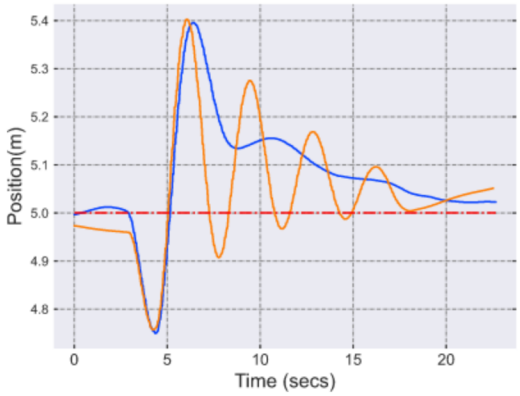
For visual tracking, an additional PID controller was included into the outer loop of the cascaded control, due to the instability of the system during the visual tracking tests. K_p, K_i, and K_d were set to be 0.3, 1E-5, and 1E-5 respectively to the the proportional, integral, and derivative gains of the system. Plots for the stationary target tracking and disturbance tracking are shown in Figure 41.

For the stationary target tracking, the cascade controller did not overshoot but had undershoot, to prevent the system from reaching instability from the visual tracking and referencing. The LQ controller had 2 peak overshoots, with the maximum percent overshoot at 8%. For rise time, the cascade control reached the 90% threshold in 7.1 seconds, while the LQ controller's rise time was 2.81 seconds. For settling time, the cascade control system took 13.1 seconds, while the LQ controller was able to settle in 8.23 seconds.

For the wind disturbance test, it can be seen that the cascade and LQ controller both reach stability in 15 seconds. Qualitatively, it can be noticed that cascade control has oscillations in position control while the LQ controller has a transient response. Figure 42, displays the results of the system tracking a moving target. For cascade control, the system once again undershoots,to prevent the system from reaching instability while the LQ controller has a percent overshoot of 4%. For rise time, the cascade controller reached the



(a) Stationary Visual Target Tracking



(b) Wind Disturbance Visual Tracking

Figure 41: Tracking Stationary Target and Disturbance Tracking

90% threshold at 8.34 seconds, while the LQ controller's rise time was 6.125 seconds. For settling time the cascade controller took 13.6 seconds, while the LQ controller took 7.6 seconds to reach steady state. Results from these 4 tests conducted showcase the robustness of the LQ feedforward controller. The nominal position reference tracking, identifies the LQ's feedforward controller ability to reach a steady state response in a faster manner than the cascade control law as well as have no overshoot. However what is very important to look at are the results of the vision tracking of the control law. Although the LQ system has overshoot in the visual tracking, it is able to regulate and drive the overshoot down in a transient manner, without frequent oscillations. This is highly desirable since the vision tracking of the fiducial tag's become much more inaccurate with aggressive changes of attitude and position to the system, inducing poor tracking of the quadcopter.

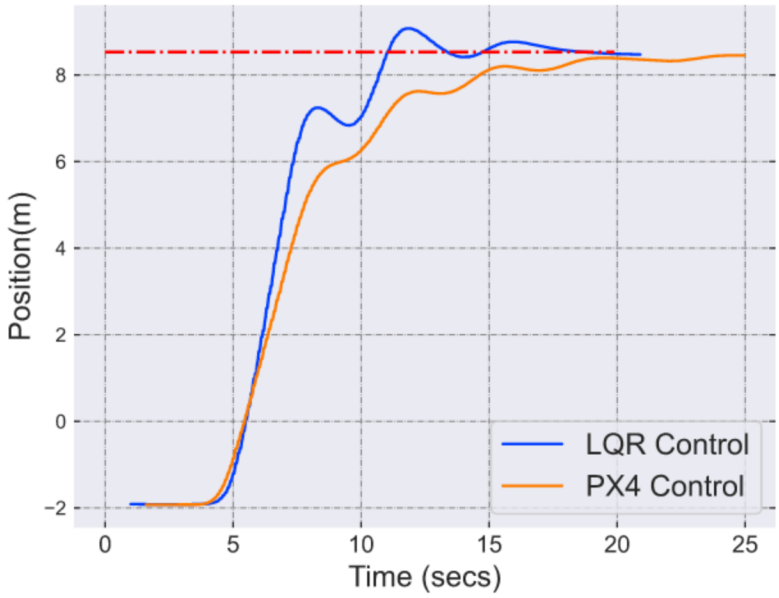


Figure 42: Moving Target Tracking

5.10 Precision Landing with a Gusto

With the results of previous tests, a final test was conducted, the precision landing of the quadcopter system with the LQG. During the precision landing from 15m in altitude, disturbance gust winds of 9.5 m/s from the 8 cardinal directions, lasting 1.5 seconds, were injected in the simulation during the precision landing operation of the quadcopter. This was done to verify the robustness of the system when the operation was conducted. Figure 43 shows the images of the simulation and operation of the precision landing operation. As the system is subject to disturbances as shown in Figure 43b and 43c, the LQ control system is robust enough to overcome the disturbances applied to the

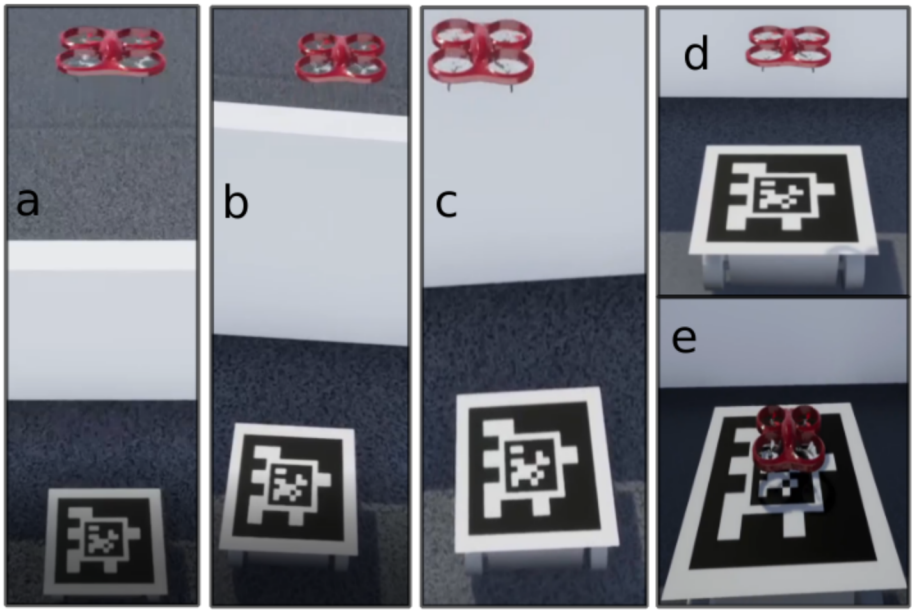


Figure 43: Precision Landing in Simulation

quadcopter, stabilize itself, and track the fiducial tag and conduct proper landing protocols, displayed in Figures 43d and 43e. Figure 44 displays the flight trajectory of the quadcopter during the disturbance inputs, the final position of the quadcopter was 5.03m and 5.02m in the x and y body frames of the quadcopter. Which is 0.03m and 0.02 meters off from the true center of the fiducial tag.

5.11 Conclusion

In this chapter, an LQ feed forward precision tracking and landing guidance and control law for fiducial tags was developed for a quadcopter system for UTM missions. This guidance and control law incorporates a Kalman Filter that observes and tracks the fiducial tag's from a corrected distortion calibration. Tests were conducted to compare the LQ feed forward controller and the PX4 cascaded controller, where the LQ proved

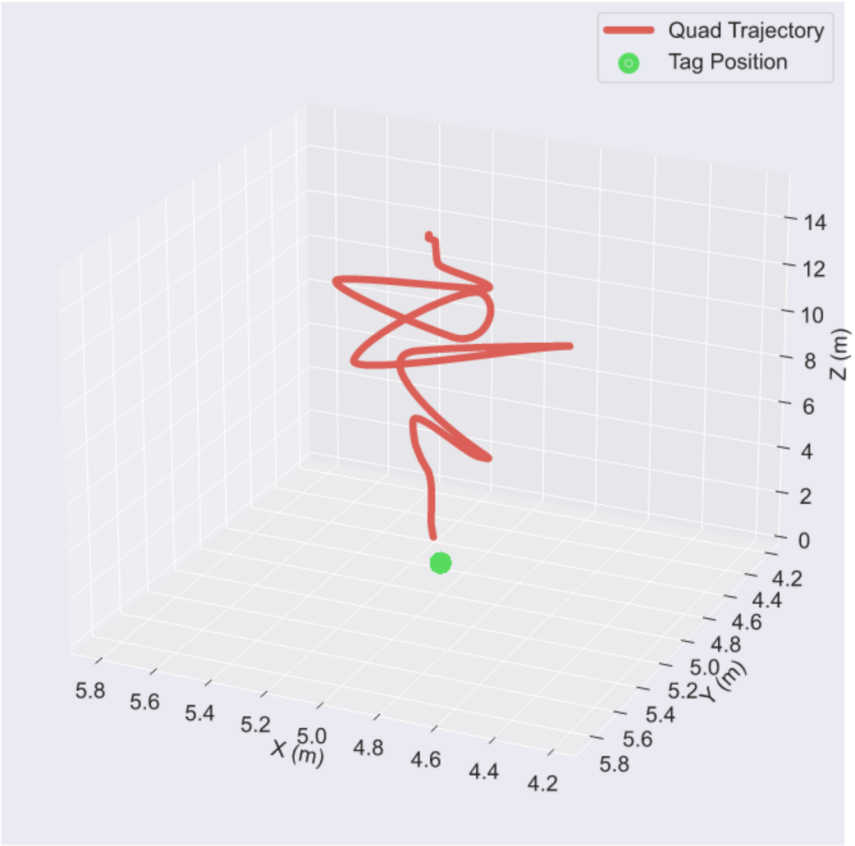


Figure 44: Quadcopter Trajectory during Precision Landing from Disturbances

to be robust in handling gust disturbances applied to the quadcopter as well as providing adequate settling and rise times for tracking the fiducial tags.

CHAPTER 6

OVERALL CONCLUSION AND FUTURE WORK

In this thesis a High Fidelity simulation was proposed and implemented for testing UAS behaviors in UTM. The development of this simulation was done to allow future interaction and testing of any USS Service provided for UAS. This simulation provides high fidelity visualization of UAS operations, the ability to collect quantitative data, and implement actual protocols of UTM with the addition of database and sharing of information. Although the simulation provides a depth of realism, the computational and graphical costs demands a powerful computer to operate, and thus lower-end computers will not be sufficient to operate with.

In addition, this thesis contributes a multi-UAS path-finding algorithm that has been tested utilizing a Low Fidelity Monte-Carlo and High Fidelity simulation previously mentioned. The algorithm provided allows ease of scalibility for multiple UAS operations as well as ease of integration The algorithm's results provides favorable results in that for operations under 50 UAS, success rate was above 99% and that the time to plan solutions for 100 UAS on average takes less than 50 seconds. It must be noted that the algorithm currently proposed is not real time based and does not consider time. Currently this algorithm would be best suited for pre-depature of UAS.

Last, this thesis provides a Linear Quadratic Gaussian control for tracking and

landing missions of an AprilTag with a feedforward controller. Synthesis of the development of the Kalman Filter and the Linear Quadratic feed forward controller are discussed. Simulation tests were conducted utilizing the simulation framework, with disturbances of wind applied to the quadcopter and noise added onto the computer vision to test robustness of the system. Results were favorable for the control method in comparison to a cascade controller.

For future work of the simulation, it would be desirable for the framework of the High Fidelity simulation to migrate to ROS 2, where there is no longer any need of a ROS master node to conduct any operations. This would allow additional depth of simulating individual agents and extensibility to allow different port connections to simulate services at different regions.

For the future work on the algorithm incorporating a different search methods such as a Genetic-Algorithm(GA) for the high-level search method is considered. Furthermore, incorporating 4 dimensions (x, y, z, t) instead of 3 dimensions (x, y, z) in the reservation table would allow approximation of what waypoints and corridors are vacant based on not just position but time as well. Last coupling the path-planning algorithm with real-time path planning methods and with perhaps a decentralized method for this search. This would help reduce any probability of collision from occurring between UAS as well as any random obstacle during real-time flight operations.

For future work of the control law, further tuning of the system methods for the LQG to find the best performance characteristics for tracking and landing missions. In

addition live tests are desired to be performed with an actual quadcopter to have comparisons between the simulated and actual results for future post-processing of results.

REFERENCE LIST

[1] Concept of Operations V2.0. *FAA 2*, 1 (2015), 460, 466.

[2] Abbas, S. M., Aslam, S., Berns, K., and Muhammad, A. Analysis and Improvements in AprilTag Based State Estimation. *Sensors 19*, 24 (2019).

[3] Bernhardsson, B. Control System Design - LQG, August.

[4] Bertozzi, M., Broggi, A., Fascioli, A., Tibaldi, A., Chapuis, R., and Chausse, F. Pedestrian localization and tracking system with Kalman filtering. In *IEEE Intelligent Vehicles Symposium, 2004* (2004), pp. 584–589.

[5] Botea, A., Müller, M., and Schaeffer, J. Near optimal hierarchical path-finding (HPA*). *Journal of Game Development 1* (01 2004).

[6] Chen, S. Y. Kalman Filter for Robot Vision: A Survey. *IEEE Transactions on Industrial Electronics 59*, 11 (2012), 4409–4420.

[7] Dario Floreanol, R. J. W. Science, technology and the future of small autonomous drones. *Nature 521*, 463 (2015), 460, 466.

[8] Desaraju, V. R., and How, J. P. Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees. 4956–4961.

[9] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. CARLA: An Open Urban Driving Simulator, 2017.

[10] Doyle, J. Guaranteed margins for LQG regulators. *IEEE Transactions on Automatic Control 23*, 4 (1978), 756–757.

[11] Gunjal, P. R., Gunjal, B. R., Shinde, H. A., Vanam, S. M., and Aher, S. S. Moving Object Tracking Using Kalman Filter. In *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)* (2018), pp. 544–547.

[12] Guo, Y., Guo, J., Liu, C., Xiong, H., Chai, L., and He, D. Precision Landing Test and Simulation of the Agricultural UAV on Apron. *Sensors 20* (06 2020), 3369.

[13] Hart, P. E., Nilsson, N. J., and Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.

[14] Hoekstra, J., and Ellerbroek, J. BlueSky ATC Simulator Project: An Open Data and Open Source Approach.

[15] Hofstede, G. J., de Caluwe, L., and Peters, V. Why Simulation Games Work-In Search of the Active Substance: A Synthesis. *Simulation and Gaming 41 (2010) 6 41* (01 2010).

[16] Jaewoo Jung, Joseph Rios, M. X. J. H. P. L. Overview of NASA's Extensible Traffic Management (xTM) Research, Jan. 2020.

[17] Justin Nguyen, P. K. N., and Abdulrahim, M. Development of an Unmanned Traffic Management Simulation with Robot Operating System and Gazebo. *AIAA SciTech AIAA 2022-1918* (2022).

[18] Koenig, N., and Howard, A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)* (2004), vol. 3, pp. 2149–2154 vol.3.

[19] Koks, D. Using Rotations to Build Aerospace Coordinate Systems.

[20] Lavretsky, E. *Optimal Control and the Linear Quadratic Regulator*. Springer, 2012.

[21] Liang, X., Chen, G., Zhao, S., and Xiu, Y. Moving target tracking method for unmanned aerial vehicle/unmanned ground vehicle heterogeneous system based on AprilTags. *Measurement and Control 53* (01 2020), 002029401988907.

[22] Liu, W., Zheng, Z., and Cai, K.-Y. Bi-level programming based real-time path planning for unmanned aerial vehicles. *Knowledge-Based Systems 44* (2013), 34–47.

[23] Marcus Johnson, J. L. UAS Traffic Management Conflict Management Model, Jan. 2020.

[24] Millan-Romera, J., Acevedo, J., Rodríguez Castaño, , Perez-Leon, H., Capitán, C., and Ollero, A. A UTM simulator based on ROS and Gazebo. pp. 132–141.

[25] Millan-Romera, J. A., Perez-Leon, H., Castillejo-Calle, A., Maza, I., and Ollero, A. ROS-MAGNA, a ROS-based framework for the definition and management of

multi-UAS cooperative missions. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)* (2019), pp. 1477–1486.

[26] Olson, E. AprilTag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation* (2011), pp. 3400–3407.

[27] Papanikolopoulos, N., Khosla, P., and Kanade, T. Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision. *Robotics and Automation, IEEE Transactions on 9* (03 1993), 14 – 35.

[28] Pei, Y., Biswas, S., Fussell, D., and Pingali, K. An Elementary Introduction to Kalman Filtering. *Communications of the ACM 62* (10 2017).

[29] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. ROS: an open-source Robot Operating System. vol. 3.

[30] Rios, L. H. O., and Chaimowicz, L. A Survey and Classification of A* Based Best-First Heuristic Search Algorithms.

[31] Roberge, V., Tarbouchi, M., and Labonte, G. Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning. *IEEE Transactions on Industrial Informatics 9*, 1 (2013), 132–141.

[32] Saraf, P., Gupta, M., and Parimi, A. M. A Comparative Study Between a Classical and Optimal Controller for a Quadrotor. In *2020 IEEE 17th India Council International Conference (INDICON)* (2020), pp. 1–6.

[33] Shah, S., Dey, D., Lovett, C., and Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics* (2017).

[34] Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence 219* (2015), 40–66.

[35] Silver, D. Cooperative Pathfinding. *Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2005* (01 2005), 117–122.

[36] Sturm, P. *Pinhole Camera Model*. Springer US, Boston, MA, 2014, pp. 610–613.

[37] Tahir, Z., Jamil, M., Liaqat, S., Mubarak, L., Tahir, W., and Gilani, S. State Space System Modeling of a Quad Copter UAV. *Indian Journal of Science and Technology 9* (07 2016).

[38] Tosun, C., Işık, Y., and Korul, H. Comparison of PID and LQR controllers on a quadrotor helicopter Demet.

[39] Veeraraghavan, H., Masoud, O., and Papanikolopoulos, N. Computer vision algorithms for intersection monitoring. *IEEE Transactions on Intelligent Transportation Systems 4*, 2 (2003), 78–89.

[40] Wang, P., Man, Z., Cao, Z., Zheng, J., and Zhao, Y. Dynamics modelling and linear control of quadcopter. In *2016 International Conference on Advanced Mechatronic Systems (ICAMechS)* (2016), pp. 498–503.

[41] Yao, P., Wang, H., and Su, Z. Real-time path planning of unmanned aerial vehicle for target tracking and obstacle avoidance in complex dynamic environment. *Aerospace Science and Technology 47* (2015), 269–279.

[42] Zammit, C., and Kampen, E.-J. V. *Comparison between A* and RRT Algorithms for UAV Path Planning*.

[43] Zhang, Z. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 11 (2000), 1330–1334.

VITA

Justin Nguyen was born on July 12, 1994 in Sikestone,MO. Justin Nguyenwas educated in various locations around Kansas and Missouri and graduated from William Chrisman in 2012. After realizing being an architect was not what we he wanted to do because he can't cut straight lines, Justin Nguyendecided to join the Missouri National Guard as a Combat Engineer and after coming back decided to go back to school for Engineering in 2018. Justin Nguyengraduated with his Bachelor's in 2020 and plans on pursuing his phD with Dr. Mujahid Abdulrahim, with an emphasis on Computer Science.

Figure 45: How Justin Feels Right Now, Shoutout to Simeon and Ethan for this comic