Autonomous Flight Control System for Drones to Reach Targets

Adrian Abedon
Department of Computer Science
Carnegie Mellon University
aabedon@andrew.cmu.edu

December 10, 2020

Abstract

We investigate quadcopter drones as a cyber-physical system and develop a controller that navigates the drone to various goals in 3D space. The proposed controller has three components. The first is an acceleration controller that is verified in KeYMaeraX, the second converts the acceleration to the desired orientation and altitude for the third component, a PID controller. The controller is implemented in Python and runs on a drone simulated in AirSim, an open source simulator built on Unreal Engine. An environment is constructed with several goals for the drone to navigate. The simulation confirms that the proposed controller works well and can be used in real drones.

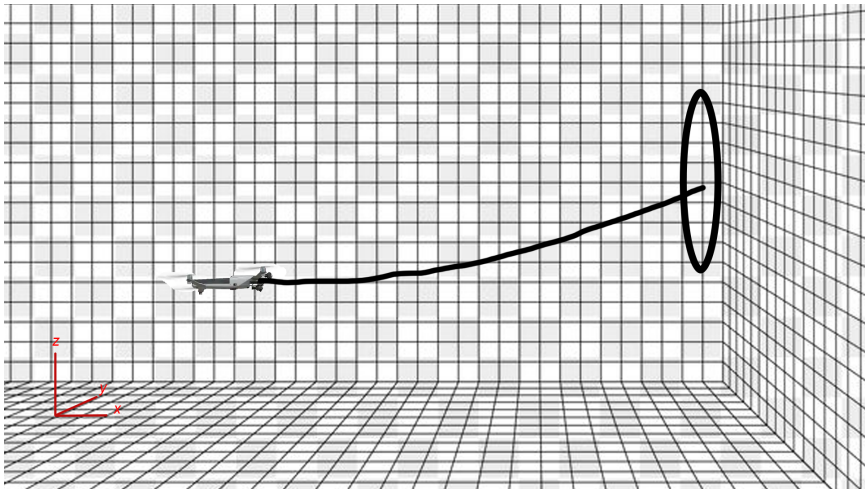


Figure 1: Drone navigating to a goal

1 Introduction

Unmanned aerial objects (UAVs) have become important in the realm of cyber-physical systems as they proliferate into many aspects of our lives from autonomous drone delivery, emergency response, security surveillance, and more. Ensuring that UAVs can safely navigate the world around them is a critical component of their physical and software design. UAV's are often given the task of navigating to some location in 3D space safely. In this project, we investigate quadcopter drones as a cyber-physical system and create a controller that can safely and efficiently navigate the quadcopter through a hoop in a 3D environment (Figure 1). The controller will have three components. The first is a time-triggered controller verified in KeYmaeraX that gives desired acceleration values along the 3D axes to reach the goal. The second component converts the desired acceleration values to command the roll, pitch, yaw, and altitude of the quadcopter. The third component is a PID controller that determines the power delivered to the quadcopters motors to obtain the desired orientation and altitude through space. The three components of the control are implemented in Python to communicate with the open source drone simulator AirSim. The results of the simulation are analyzed to determine the success of the controller.

2 Related Work

The traditional design of a PID controller for the roll, pitch, yaw, and altitude of a quadcopter is well established as it is the most common control scheme for consumer drones [1]. Variations on the traditional quadcopter PID controller have been explored by using a PID controller on the position and heading of the drone (x, y, z, ψ) in order to follow a predetermined path in 3D space [2]. Additionally, there has been research into the combination of PID control with fuzzy logic to further optimize the performance of a quadcopter controller [3, 9]. Our control scheme differs from the above as it combines a verified controller with a PID controller to navigate the 3D world.

Drones usually require the ability to sense the world around them and recognize obstacles and targets. There has recently been an abundance of research in autonomous drone navigation that focus on obstacle recognition through computer vision systems [4, 5]. Many of the controllers rely heavily on machine learning or its subpart deep reinforcement learning to determine the position of the drone and obstacles. While this research is critical to the deployment of drones in the real world, the underlying controllers that they are using are not verified for safety. Machine learning is lauded for its ability to deal with unpredictable scenarios but can themselves be unpredictable. For this reason, we omit the use of these algorithms and instead elect to simplify our testing by assuming the drone has immediate access to its position, velocity, and target position.

Another important aspect of cyber physical systems research around UAV's is the real-time verification of flight control systems [6]. As control schemes become more complicated and difficult to verify, it is important to have systems in place that check if the controller is acting unsafely at a given time. While this project does not use real-time verification for the proposed

controller, we investigate its performance and safety in real time in order to make improvements.

3 The Quadcopter Model

The model that we consider is a quadcopter with four rotors that can individually apply an upward thrust. The rotation direction of two of the motors are clockwise while the other two rotate counterclockwise. This allows for the angular momentum of the motors to cancel out and produce yaw motions as needed (Fig 2). With four motors, a quadcopter has independent control over four degrees of its motion, its altitude, roll, pitch, and yaw.

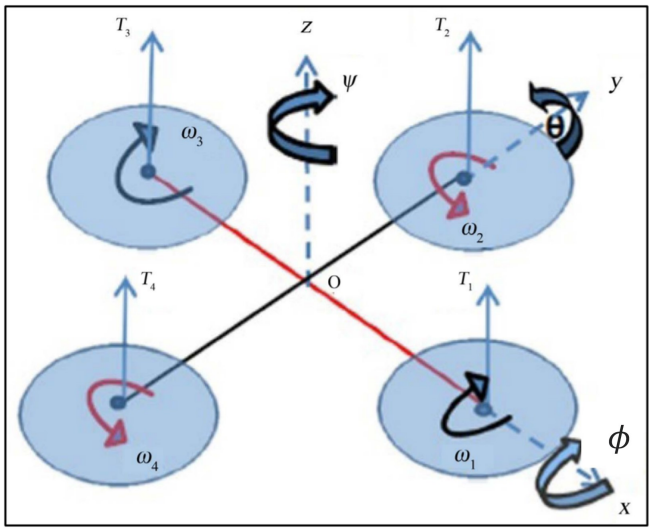


Figure 2: Diagram of a quadcopter's important variables. Here, ϕ is the roll, θ is the pitch, and ψ is the yaw.

By controlling the roll and pitch, a quadcopter can also move laterally adding 2 additional degrees of motion. This means that a quadcopter is an underactuated system with 4 input forces and 6 output states $(x, y, z, \phi, \theta, \psi,)$ where ϕ is the roll, θ is the pitch, and ψ is the yaw as shown in Figure 2. Since a quadcopter can independently control its altitude, roll, and pitch, it therefore has independent control over its x, y, and z movement. The z movement is connected to the altitude control, and the x and y movement are connected to the pitch and roll of the quadcopter respectively. Additionally, quadcopters can be actuated very quickly, resulting in very precise and immediate control of the roll and pitch. Thus, when developing the verified controller for the drone in KeYmaeraX, we can assume that the drone has immediate control over its acceleration in the x, y, and z direction independently. We also assume that there is no air resistance which is a good assumption for the drone at low speeds where the controller is intended to operate. Further assumptions that we make in our model is that the structure of the quadcopter is fully rigid, so the thrust of each motor is applied in parallel. Additionally, we assume the structure of the quadcopter is symmetrical along each axis defined

in Figure 1, implying that no motor lies farther from the center of mass than another allowing for each to apply an equivalent torque to the frame. Finally, to simplify control, we omit modeling turbulence.

4 Creating a Controller

4.1 Structure of the Controller

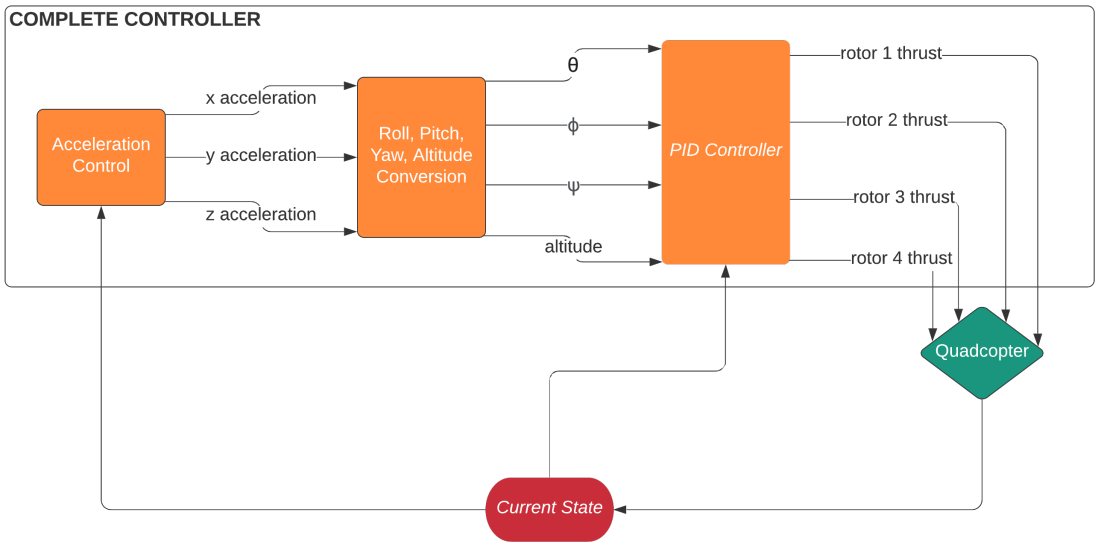


Figure 3: Structure of the controller

The controller that we designed for the drone has three separate components that feed data into one another as shown in Figure 3. The components works together as follows. First, the current state of the world and drone is read through the sensors. This state information is fed to a controller that determines the acceleration values along the x, y, and z axes that are required to reach the goal hoop. Then, the accelerations along the axes are are fed to the next component of the control which determines the roll, pitch, yaw, and altitude that the drone should achieve by the next control cycle. Finally, the roll, pitch, yaw, and altitude are inputted into a PID controller which sends thrust commands to the rotors on the quadcopter to obtain the desired state. The PID controller updates the rotor thrust at a much faster rate than the acceleration controller sends commands. Therefore, in-between when the acceleration controller sends commands, the PID controller is reading the sensor data independently.

4.2 Formally Verified Acceleration Control

The first layer of control for the quadcopter was implemented in differential dynamic logic (dL) and verified in KeYmaeraX [10]. This controller's job is to provide the acceleration of the quadcopter along each axis such that it passes through a goal hoop safely as shown in Figure 1.

4.2.1 Constants and Variables

We make the assumption that the controller has access to the drone's location, the goal's location, and the drone's velocity. This is a reasonable assumption to make since position can be determined by GPS systems on a real drone and velocity can be estimated from that data as well as the gyroscopic sensors. The constants we use in the model are:

- (gx, gy, gz) - The location of the goal
- $grav$ - Acceleration due to gravity
- Az - Maximum acceleration along the z-axis
- Ay - Maximum acceleration along the y-axis
- T - Time between each acceleration control

The variables we use in the model are:

- (x, y, z) - The position of the drone
- (vx, vy, vz) - The velocity of the drone
- (ax, ay, az) - The acceleration of the drone
- t - Elapsed time since the last control

4.2.2 Assumptions

For our model we make the additional assumptions that the hoop is located on the yz-plane and that the drone's x, y, and z movement is aligned along the coordinate system as shown in Figure 1. When the drone moves forward on the x-axis, it is moving perpendicular to the yz-plane defined by the hoop. We can make this assumption because we are able to define the world coordinates so that the hoop lies in the yz-plane and have the quadcopters coordinates match since it has independent control of its movement along all axes.

4.2.3 Preconditions

Before the acceleration controller begins, the following conditions must be met.

- The drone starts at position $x = 0$.
- The goal lies in front of the drone, $gx > x$.
- The drone is moving towards the goal, $vx > 0$. This condition is used to simplify the model. It is a reasonable simplification since a constant positive x velocity can be induced by simply commanding the PID controller to tilt the drone forward slightly.
- The drone's maximum acceleration along the z-axis must be able to overcome gravity, $Az > grav$, which is positive, $grav > 0$.
- The drone must have the ability to move along the the ability to move along the x and y axes, $Ax > 0$ and $y > 0$.
- The acceleration required along each of the axis must fall within the acceleration bounds. The derivation of the following equation will be explained in section 4.2.6.

$$-Ay \leq (2 \cdot ((gy - y) - vy \cdot T_L)/T_L^2 \leq Ay \tag{1}$$

$$0 \leq (2 \cdot ((gz - z) - vz \cdot T_L))/T_L^2 + G \leq Az \tag{2}$$

4.2.4 Dynamics

The differential equation that models the dynamics of the system assumes no air resistance or turbulence, and direct control over acceleration along the axes.

$$\{x' = vx, y' = vy, z' = vz, vy' = ay, vz' = az - grav\}$$

Here we assume that there is no acceleration along the x-axis since quadcopter will move with a constant positive velocity along the axis. The position and velocity of the drone are modeled by simple Newtonian physics in the acceleration controller. Although in the real world there exists air resistance and turbulence, we can make these simplifications since the drone will be performing low speed maneuvers and the PID controller can compensate for discrepancies in the desired acceleration and the actual acceleration.

4.2.5 Reaching the Goal

We can then define the properties of the system which accurately models the safety, efficiency, and success of the quadcopter traveling through the hoop as indicated below:

- The success of the drone flying through the hoop can be defined as when the drone enters the hoop's yz-plane, it must be at the center of the hoop.

$$x = gx \implies (y = gy \wedge z = gz) \tag{3}$$

- We also want the drone to always be moving towards the hoop until it passes through it, guaranteeing that the model will not get stuck.

$$x \leq gx \implies vx > 0 \tag{4}$$

- The drone's acceleration along each axis must also not exceed the defined Ay and Az values. Here Ay constrains the maximum acceleration in either direction along the y-axis. Az constrains the maximum upward acceleration applied by the motors along the z-axis. The downward acceleration applied by rotors must be greater than 0 and occurs when they are not spinning.

$$-Ay \leq ay \land ay \leq Ay \land 0 \leq az \land az \leq Az \tag{5}$$

4.2.6 Control

In designing a controller for the drone, we assume that the drone moves at a constant velocity towards the hoop. With a constant forward velocity, the model guarantees that the drone does not get stuck in front of the hoop and fail to cross its yz-plane. Additionally, in the real world, a constant forward velocity can be induced by simply commanding the PID controller to tilt the drone forward by a small amount, where air drag will limit the velocity to a constant value. Since we assume that the drone moves towards the hoop with constant velocity, we can easily calculate the time left until the drone intersects the hoop's yz-plane as:

$$T_L = \frac{gx - x}{vx}$$

Thus, in T_L time, the drone's y position must match the goal's y position. This is modeled by the equation:

$$gy = y + vy \cdot T_L + \frac{ay}{2} \cdot T_L^2 \tag{6}$$

The controller can then set the desired acceleration along the x-axis by solving for ay.

$$ay = \frac{2\left((gy - y) - vy \cdot T_L\right)}{{T_L}^2} \tag{7}$$

The acceleration along the y-axis applied by the motors can be determined similarly with $grav$ added since the thrust must overcome the force of gravity to accelerate upwards.

$$az = \frac{2\left((gz - z) - vz \cdot T_L\right)}{{T_L}^2} + grav \tag{8}$$

4.2.7 Proof

A proof of the acceleration controller that follows the proposed model is relatively straight forward. The hybrid program was written in dL and the proof was completed in KeYmaeraX. To complete the proof we used induction to prove loop invariants, which then implied the post condition. The first loop invariant is that when the drone enters the goal's yz-plane, it must be at the center of the goal (Eqn. 3). The next invariant is that the x velocity is always positive, $vx > 0$. Additionally, the drone must be able to accelerate towards the goal within the defined acceleration bounds as modeled by equations 1 and 2. Finally, the current acceleration along the y and z axis must be within bounds, as modeled by Equation 5. All of the loop invariants were proved using the solve tactic for the ODE and then quantifier elimination. Proving that the precondition implies the loop invariant and the loop invariant implies the post condition was completed with quantifier elimination as well. The proof is attached in the project files.

4.3 Roll, Pitch, Yaw, Altitude Control

The next component of the control is converting the acceleration values, outputted by the verified controller, to the desired roll, pitch, yaw, and altitude values for the PID controller. We will define values which we can use as a starting point. Since the quadcopter should not rotate about the z-axis through its control, yaw should remain zero:

$$\psi = 0 \tag{9}$$

The roll and pitch of the quadcopter are a function of the upward acceleration of the quadcopter and the acceleration along the x and y axes. Since the acceleration along the x-axis is zero while maintaining a constant x velocity, the pitch (rotation about the y-axis) should be zero. In reality, however, maintaining constant velocity requires a small tilt, ϵ_θ, of the drone about the y-axis to overcome air resistance.

$$\theta = 0 + \epsilon_\theta \tag{10}$$

The roll of the drone (rotation about the x axis) will increase as the acceleration along the y-axis increases and decrease as the upward acceleration increases according to the following equation:

$$\phi = \tan\left(\frac{ay}{az}\right) \tag{11}$$

Finally, the desired altitude of the PID controller is related to the update time of the control T. We want the drone to move in time T towards a future altitude that guarantees that it will be able to reach the center of the hoop as indicated below:

$$altitude = z + vz * T + \frac{1}{2}(az - grav) * T^2 \tag{12}$$

Setting the roll, pitch, yaw, and altitude to these values may work in theory, however, the real world has air resistance, turbulence, and other complex interactions that were not taken into account in the verified model. With testing and simulation, we determined scale and bias values that bring these approximations closer to the desired real-world performance and is covered in section 5.3.

4.4 PID Controller

With the acceleration outputs of the verified controller and the roll, pitch, yaw, and altitude conversion, we now aim to create a PID controller which adjusts motor thrust to robustly match the desired state.

4.4.1 Explanation

A PID controller works by utilizing a closed feedback loop. Typically, there is a process variable than needs to be controlled, like the roll of the quadcopter. A sensor measures the current value of the process variable and that information is provided to the PID controller. The set point is the desired value of the process variable that controller is attempting to achieve. An algorithm,

the compensator, uses the difference of the current value and the desired value of the process variable to determine the actuation of the system, such as the thrust of each rotor. Then the system performs the actuation and the loop is restarted [8]. The actuation of the rotors is dictated by the proportional (K_p), integral (K_i), and derivative (K_d) parameters of the PID controller. Changing each parameters has both benefits and tradeoffs summarized in Figure 4.

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

Figure 4: The effects of adjusting each PID parameter

4.4.2 PID controllers Used

For our controller, we used a PID controller for the roll, pitch, yaw, and altitude of the quadcopter. We chose to use a controller on these components of the quadcopters movement since they correlate to the altitude and gyroscopic sensors typically found on drones. Additionally, over 90 percent of the quadcopter controllers in operation are PID controllers since they are viewed as simple, reliable, and easy to understand [9]. Since the focus of this project is not to implement the PID controller, we used one that comes with AirSim, an open source drone simulator built on Unreal Engine. Through experimentation, we determined the optimal K_p, K_i, and K_d parameters such that the drone responds as expected.

5 Implementation and Simulation

5.1 AirSim

AirSim is an open source simulator developed by Microsoft as a plugin for Unreal Engine. Using the Unreal Engine to render photo realistic scenes, AirSim provides the frame work for highly accurate drone simulations. The physics engine that is used by AirSim takes into acount gravity, air desity, air pressure, magnetic fields, and the kinematic state of other bodies in the simulation. The detail of the simulation differs from the simpler assumptions that were made in developing the controller. Despite these differences, the controller that we implemented still performed well in the simulated environment.

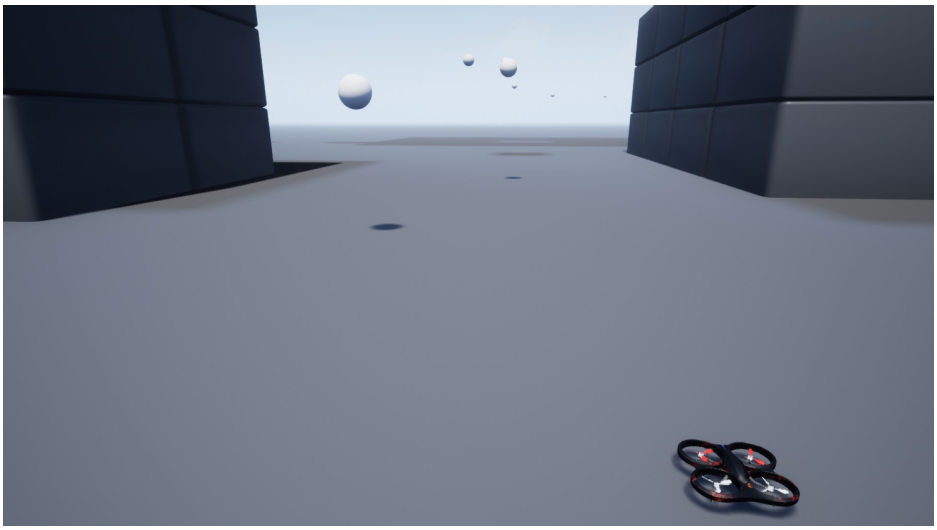


Figure 5: Course constructed in Unreal Engine with seven goals and the drone is simulated by AirSim.

Within AirSim, we set up a course that the drone must follow as shown in Figure 5. We used spheres to indicate the center of the hoops since using actual hoops in the software required modeling outside of Unreal Engine or purchasing model packs. The drone's goal is to travel through all of the spheres in one run. We used the latest version of AirSim pulled from GitHub and Unreal Engine 4.25.4.

5.2 Controller Implementation

A controller that follows the scheme outlined in section 4 was written in Python 3. Communication with the simulated drone was established through API's provided by AirSim. Before our controller begins navigation, the drone is instructed to take off and begin moving forward slowly. Then, the control begins reading the state information and sending the desired roll, pitch, yaw, and altitude to the PID controller. We used the PID controller in the *simple_flight* API provided by AirSim. There are many ways to navigate the drone using *simple_flight*, but most critically it has a PID controller for the roll, pitch, yaw, and altitude with adjustable integral, proportional, and derivative parameters. The implemented controller can be found in the project files.

5.3 Control Parameters

A critical component of creating a controller that performed well in the simulation was determining the time between each control cycle, T, and the optimal parameters, (K_p, K_i, K_d), for the PID controller. Firstly, the PID controller built into AirSim updates motor thrust by default at a rate of 1000Hz. Additionally, it operates independently from the rest of the controller in between when it is sent desired values for the roll, pitch, yaw, and altitude of the drone. What

we found while running the simulation was that decreasing the time between each control cycle T yielded poor response from the PID controller in matching the desired altitude. The altitude sent to the PID controller changes every control cycle since it is defined as the desired location of the drone in time T. Thus, we expect the PID controller to reach the altitude in time T. However, as T decreases, the difference in between the current altitude and the desired altitude sent to the controller decreases as well. Due to turbulence and the design of the PID controller, it does not do well adjusting to these small differences. To combat this trade off, the time between each acceleration control, T, was set to 3.33Hz, a much less frequent update than the PID controller.

Setting the PID controller parameters also presented its own set of challenges. The parameters for the PID controllers for the roll, pitch, and yaw remained at their default values of $K_p = 0.25$, $K_i = 0$, and $K_d = 0$. For the altitude PID controller, small variations in each of the parameters had very large affects on the result. We found that setting $K_p = 0.60$, $K_i = 0.001$, and $K_d = 0$ for the altitude PID controller yielded the best results in our simulation. These parameters are specific to the drone and would need to be adjust if a different drone was simulated. They are also specific to the control cycle time, T, since they determine how quickly and correctly the drone achieves the desired altitude. We found that with a smaller, T, we were unable to find parameters for the altitude PID controller that worked well.

Additionally, a constant forward pitch, ϵ_θ, was set to 0.03 radians. Sending this small pitch to the PID controller gave the drone a positive forward velocity throughout the simulation which matches the assumption made in the design of the controller.

Finally, we discovered that while in theory the roll defined in equation 8 should effectively control acceleration along the y-axis, in practice it needed to be scaled by some factor. We found that scaling the desired roll by $\phi_S = 1.7$ yielded the desired performance in the drone's movement. All of the adjusted parameters are summarized in Figure 6.

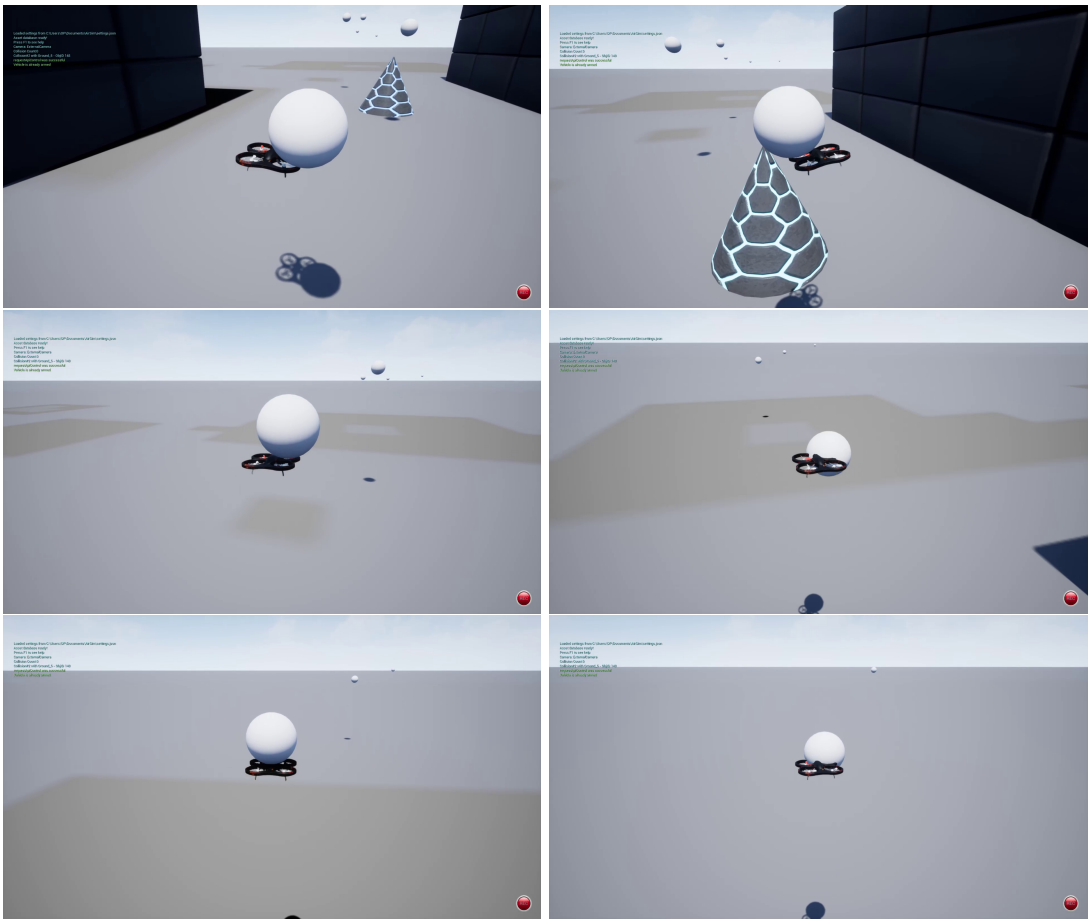
Summary of Parameters	
Parameter	**Value**
T	0.3
K_p (Z)	0.60
K_i (Z)	0.001
K_d (Z)	0
ϵ_θ	0.03
ϕ_S	1.7

Figure 6: Summary of adjusted parameters

6 Results

6.1 Demonstration

By using the implemented controller, the simulated drone was able to successfully navigate to each of the 7 goals. The distance of the drone to the perfect center of the goals was within about one drone width of the goal. Images of the drone passing each goal is shown in Figure 7.



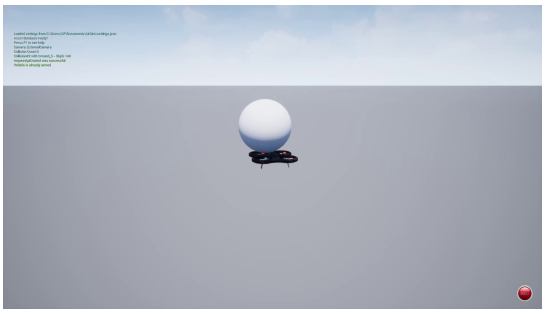


Figure 7: Clips from the simulation (read left to right followed by top to bottom).

6.2 Analysis of Velocity and Acceleration

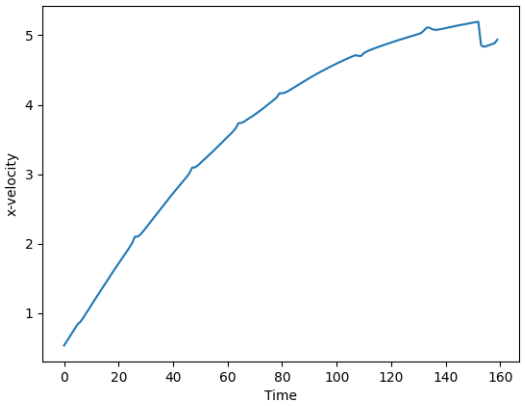


Figure 8: Graph of the velocity along the x-axis versus time. The velocity increases throughout the entire simulation which differs from our original assumptions.

By running the simulation, we found that our expectation of a constant x-velocity was not necessarily true. The velocity of the drone along the x-axis continued to increase throughout the run, only beginning to plateau at the end as shown in Figure 8. Although our assumption that the drone moves along the x-axis with constant velocity was not met, the success of the drone's navigation demonstrates that our controller is robust to error.

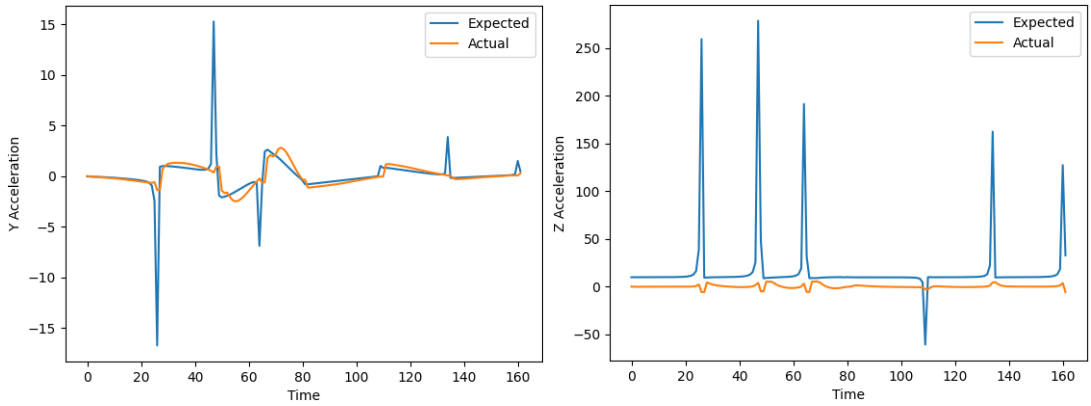


Figure 9: Graph of the expected and actual acceleration along the y-axis (left) and graph of the expected and actual acceleration along the z-axis (right).

We also found for the most part, the expected and actual acceleration along the y-axis were very similar in the simulation. The large spikes in expected acceleration in the graph (Fig. 9) occur when the drone gets close to the goal and the controller is trying to correct for the relatively small error in a very short amount of time. The drone does not have the ability to accelerate at these large values and therefore does not perfectly intersect the goal, however, it does fly closely to the target.

The graph of the expected and actual z-acceleration shows a gap between the model and simulation. While the drone does approach the desired z-position when it nears the goal, the actual acceleration does not match the expected acceleration. This deviation most likely stems from the limitations of a PID controller in commanding small differences in the altitude within the control time T.

7 Discussion

In this project, we succeeded in creating a verified controller for acceleration, and then using the output to command the yaw, pitch, roll, and altitude of a PID controller to navigate a drone to a desired goal. The accuracy of the controller in simulation confirms that a similar structure can be confidently implemented in a real world drone.

In bridging the gap between the theoretical and actual implementation of our controller and model, we found that certain parameters had to be adjusted in order for the drone to respond as expected. Once the parameters were properly set, the drone successfully matched the desired acceleration along the y-axis. While the drone ultimately matched the altitude of the goal, it failed to match the desired acceleration along the z-axis, making up most of the difference in altitude only when it was very close to the goal. The lack of accuracy in the control along the z-axis demonstrates the weakness of a PID controller in this scenario. In the future, a PID controller that takes into account the desired acceleration instead of just the altitude when actuating the rotors may perform better.

An important consideration is that the maneuvers performed by the drone during the simulation were done at relatively low speeds. With the current controller, high speed maneuvers would most likely fail due to the relatively low update on the acceleration controller. If an improved altitude controller is used, however, the update time can be decreased, and the precision of the control increased to perform high speed maneuvers. Nevertheless, the controller implemented for this project is a promising candidate for real drones that must navigate through various environments.

8 Deliverables

The deliverables for this project include:

- A model and acceleration controller in KeYmaeraX along with the proof
- An implementation of the three components of the controller in Python
- An executable application that runs the drone simulation
- A video of the successful navigation of the drone to various goals

References

[1] Salih, Atheer & Moghavvemi, Mahmoud & Mohamed, Haider & Gaeid, Khalaf. (2010). Flight PID controller design for a UAV quadrotor. Scientific research and essays. 5. 3660-3667

[2] C. N. R. Katigbak et al., "Autonomous trajectory tracking of a quadrotor UAV using PID controller," 2015 International Conference on Humanoid, Nanotechnology, Information Technology,Communication and Control, Environment and Management (HNICEM), Cebu City, 2015, pp. 1-5, doi: 10.1109/HNICEM.2015.7393247.

[3] M. Zareb, R. Ayad and W. Nouibat, "Fuzzy-PID hybrid control system to navigate an autonomous mini-Quadrotor," 3rd International Conference on Systems and Control, Algiers, 2013, pp. 906-913, doi: 10.1109/ICoSC.2013.6750965.

[4] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars and L. Van Eycken, "CNN-based single image obstacle avoidance on a quadrotor," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 6369-6374, doi: 10.1109/ICRA.2017.7989752.

[5] Sang-Yun Shin, Yong-Won Kang, Yong-Guk Kim,Reward-driven U-Net training for obstacle avoidance drone, Expert Systems with Applications, Volume 143, 2020, 113064, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2019.113064.

[6] Xu H, Wang P (2016) Real-Time Reliability Verification for UAV Flight Control System Supporting Airworthiness Certification. PLoS ONE 11(12): e0167168. https://doi.org/10.1371/journal.pone.0167168

[7] Zulu, A. and John, S. (2014) A Review of Control Algorithms for Autonomous Quadrotors. Open Journal of Applied Sciences, 4, 547-556. doi: 10.4236/ojapps.2014.414053.

[8] PID Theory Explained. NI, Mar. 2017, www.ni.com/en-us/innovations/white-papers/06/pid-theory-explained.html.

[9] Passino, Kevin M., and Stephen Yurkovich. Fuzzy Control. Library of Congress Cataloging-in-Publication Data, 1998.

[10] André Platzer. URL http://www.ls.cs.cmu.edu/KeYmaeraX/.