Design, Build and Test of an Automated Space Debris Laser Sweeper

A project presented to
The Faculty of the Department of Aerospace Engineering
San José State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Christopher Le

May 2022

approved by

Professor Long Lu
Faculty Advisor

San José State
UNIVERSITY

ABSTRACT

Development of a Space Debris Laser Sweeper

Christopher Le

As more nations and private entities expand their endeavors into space, this comes with the added risk of more miscellaneous hardware being released into orbit. While large debris like spent fuel tanks and decommissioned satellites can be tracked and avoided by other spacecraft, there are several small pieces of debris that are difficult to be picked up by passive sensors. While the impact from this debris may not be as severe, it can still damage critical components that would prove detrimental to the lifespan of the spacecraft. This project is intended to actively remove the debris from orbit by using a high-power laser to either completely evaporate the debris or launch the debris into an exit orbit to burn upon re-entry. A one-dimensional degree of freedom prototype has been constructed to analyze the dynamics and control of the system. Once the system has been validated for stability and responsiveness, additional degrees of freedom will be added. In the MATLAB Simulink diagram, the signal from the camera system will be used as the reference signal of the system in which the PID of the system will input a voltage through a transfer function to apply angular velocity to the reaction wheel. Through conservation of angular momentum, this will apply an angular velocity to the satellite. The position of the satellite will be validated by the camera system as well as Inertial Measurement Unit. Tracking space debris will require fine motors control, which can be achieved with a controller such as a Proportional Integral Derivative controller, or PID. Both the physical setup and the simulations use the same PID values to compare and validate the tracking performance. While the physical setup displayed steady-state tracking performance mirroring that of the simulations, it produced a larger overshoot than that of the simulations which could hinder initial acquisitions of the targeted space debris. This occurrence has been compared to the tracking performance of other reaction wheel satellite and benchmarked test displayed similar large spikes, meaning that the controller for the laser sweeper is comparable to that of other graduate project satellites. As the physical setup can roughly match that of the simulations, the tracking performance could be further improved by using an LQR controller as simulations show no overshoot when using the LQR controller; however, implementation of the LQR controller has not been completed due to the complexity involved. The performance gain over the PID controller is unknown, but the PID provides adequate tracking performance within the limitations of the hardware setup.

ACKNOWLEDGMENTS

I would like to thank Professor Lu for his guidance and for pushing me through tasks that would have been unimaginable to me during my courses. I would also like to thank my colleagues, Aaron Mandeville and Will Miller, for helping me with developing my transfer functions and Simulink. I would like to thank Katie Burrows for helping me with my electrical hardware. Finally, I would like to thank Angelina Resto, Amy Starnes, Mike Wong, William Crocker, Matt Malott, Karl Chvojka and Martin Hall for all the emotional support that helped me power through this project.

Table of Contents

LIST OF FIGURES

Nomenclature

Symbols	**Definition**	**Units (SI)**
$(\dot{*})$	Time Derivative of a variable	
τ_e	Torque	Foot-pound force (Newton-meters)
h	Angular Momentum	(Kilogram-meters squared per second)
J	Moment of Inertia	
ω	Angular Velocity	Radians per second
r	Distance from Center of Mass of System to Arbitrary point	Inches (Meters)
m	Mass	Pounds (Grams)
v	Linear Velocity	Miles per hour (Meters per second)
G	Transformation Matrix	
Ω	Wheel Velocity Vector	
k_t	Torque Constant	Foot-pound force per ampere (Newton-meters per ampere)
I_a	Current	Ampere
b	Viscous Friction	
E_b	Back-EMF Voltage	Volt
k_e	Back-EMF Constant	
V_a	Supplied Voltage	Volt
R	Motor Resistance	ohm
L_a	Motor Inductance	Henry
s	Laplace Transform Integrator	
$*(s)$	Laplace Transform Domain	

Chapter 1: Introduction

1.1 Motivation

In 1978, NASA scientist Donald Kessler theorized a scenario where the density of objects in low Earth orbit is high enough such that any collision between these objects would create even more debris and further increase the likelihood of subsequent collisions. This would culminate in a situation where the entire planet is surrounded by debris, making it impossible for humanity to conduct any more space missions. Even the smallest pieces of debris can cause significant damage to spacecraft as with the case with the 1983 STS-7 mission, where the Challenger Space Shuttle was hit by a 0.2mm paint flake or metal fragment that created a 0.4 mm diameter pit in the borosilicate window [1]. This issue is further exacerbated by anti-satellite tests where militaries would launch missiles towards satellites to test their intercept capabilities and shatter them into hundreds of pieces that spread throughout Earth's orbit. Depending on the angle and altitude that these tests take place in, most of the fragments produced by these tests will have relatively short orbit lifespans before reentering Earth's atmosphere [2]. Even still, this adds to the growing list of space debris and is detrimental to future space endeavors.

1.2 Literature Review

This section will cover literature regarding the background of orbital debris and various proposed methods to eliminate them. While there have been measures drafted for the active removal of space debris, designing space missions from producing debris is more effective for the long-term mitigation of space debris. Although NASA requires that their missions minimize debris [3], non-government agencies have only recently had preventive measures established by the Federal Communication Commission to minimize debris production [4]. According to Space Mission Analysis and Design by Wiley Larson and James Wertz, launch processes produce an average of three large trackable debris [5]. These include protective shrouds, separation devices, and expended rocket bodies. Even after insertion into the final orbit, spacecraft must deploy their tools and instruments, which may release miscellaneous hardware into space. Once the spacecraft reaches the end of its lifecycle, it should be considered debris and must be removed by propulsive maneuvers to put the spacecraft into a disposal orbit or re-entry trajectory. Equipping new satellites with an end-of-life de-orbit and orbital lifetime reduction capability is the most effective method for minimizing the number of debris produced. While this process minimizes space debris production, there are still elements in orbit that could pose a threat to spacecraft in the future which would necessitate the use of active debris removal methods.

There will still be occasions where incoming collisions could happen, and mission designers should take into consideration how the spacecraft will react accordingly. As space debris comes in all shapes and sizes, these plans will vary from energy-intensive maneuvers to passive shielding. NASA and the DOD's Space Surveillance Network cooperate to characterize satellite environments using special ground-based sensors and inspection of returning satellites. As of the time of writing this report, there are approximately 27,000 officially cataloged objects in orbit that are at least 10 centimeters or 4 inches in diameter [6]. For these objects to be considered an imminent threat to manned spacecraft like the International Space Station, they would have to have to be in the range of an imaginary box drawn around the spacecraft. This box would measure 30 miles across by 30 miles long by 2.5 miles deep and should a tracked object pass close enough for concern, Mission Control Centers in Houston and Moscow work together to develop a course of action. When an encounter is known hours in advance, a slight evasive maneuver can be performed, known as a “debris avoidance maneuver”. However, this can come at the cost of propellant and/or power, which may decrease the lifespan of the mission or direct the spacecraft into an unideal orbit. If the tracking data is not precise enough to warrant a maneuver or if the close pass isn’t identified in time to make the maneuver, the best course of action for manned spacecraft is to move the crew of the station into a spacecraft to protect them in case of loss of pressure and life support systems.

1.2.1 Mitigation Methods

These precautions primarily account for trackable space debris; untrackable pieces of debris measuring less than 10 centimeters are much more difficult to be picked up by ground-based sensors. NASA estimates that there are at least half a million pieces of debris measuring 1 centimeter and at least 100 million pieces of debris measuring 1 millimeter. While the impact caused by these pieces may not be as significant as their larger counterparts, they can still cause considerable damage, especially if vital components are struck. This is where passive protection like Whipple shields offers the greatest benefit as spacecraft do not have to actively look for space debris and perform preemptive maneuvers, potentially saving resources and offering sufficient protection. Whipple shields are thin shields set a certain distance away from the main spacecraft that are designed to break up incoming particles and disperse them over a larger area [7]. Some Whipple shield designs have fillings to minimize penetration. Despite this advanced shielding, it may be not compatible with all systems due to added weight and increased volume, thus affecting the restraints of the mission. It also does nothing to actively remove debris from the space environment as it only reduces the chances of a critical system failure.

Several initiatives have been proposed to eliminate space debris. In particular, the Japanese Aerospace Exploration Agency (JAXA) is exploring options to actively remove debris from the space environment in their Commercial Removal of Debris Demonstration program. This program is split into two phases; the first of which is Key Technology Demonstration which involves using satellites to survey debris and obtain its characteristics e.g., rotational motion or surface damage [8]. Phase II of this program is focused on Active Debris Removal but has yet to be finalized. Due to the variety of space debris, there are several methods of capturing, which are generally broken up into two categories: contact and contactless. As contactless capture methods, such as Electrostatic Tractor or Gravity Tractor, are primarily considered for asteroid deflection, they may not be viable options for debris removal systems [9]. The contact-based removal methods often involve a system or arms or entanglement devices for securing the debris. While these might be effective solutions for securing depleted thrusters or decommissioned satellites, they’re ineffective and unfeasible for smaller pieces of debris as the likelihood of successfully capturing them is much slimmer while offering a lower debris yield.

High-energy lasers may be the most feasible way to actively remove small debris measuring from 1 to 10 centimeters from space. There are two methods in which the laser would be used to eliminate debris: Direct Ablation and Ablation Back-jet [10]. Direct Ablation primarily targets debris up to 1 centimeter in length and completely incinerates it. The process of Ablation Back-jet involves targeting debris of more than 1 centimeter in length with a high-energy laser beam. The point of impact would superheat the material, causing an increase in pressure at the afflicted area. This pressure would expand and exert a reaction force on the rest of the material that would propel the debris into an elliptical orbit trajectory. At the closest point of the orbit to the earth, also known as the perigee, if the debris were to be at an altitude less than 200 kilometers, it would fall back into the earth's atmosphere where it will burn up upon reentry.

Laser systems also vary in operation; of which, there are four kinds: Solid-state lasers, Liquid lasers, Gas lasers, and Semiconductor lasers. Semiconductor lasers are the most abundant in everyday life and operate by an electrical current being reflected back and forth between two mirrors, leaving only a minor gap. The movement of this current across a Positive-Negative Junction generates light that escapes through the hole that can be further focused with lenses. Solid-State lasers operate on the same principle except that Solid-State immediately introduces ions into a host material such as glass or crystal in a flash tube, which shines light inside the system. Two reflective surfaces are placed at the end with only a small opening at one end. The light generated inside would reflect back and forth and filter out impurities until all that emitted is a uniform beam. Gas and Liquid lasers function the same way as Solid-State, the only difference being the medium used in the system. To eliminate small debris, the laser system would have to be pulsed as a pulsed laser can deliver its peak energy in an incredibly short time frame, evaporating specific parts of a debris piece without heating the entire body. The characteristics of the debris are just as important, if not more, than the laser itself. The most common materials that are found in orbital debris and micrometeoroids are Aluminum, Aluminum Oxide, Steel, and paint chips. These materials, in their various alloys, are the most common engineering materials used in space applications, hence their prevalence [6]. While not all paint used for space application is the same, common coatings are often composed of a borosilicate and aluminum mixture [11].

1.2.2 Disturbance and Noise Filtering

For adjusting the orientation of the system, reaction wheels are preferable for small satellites with long life cycles as the only resource required to operate them is energy from batteries which can also be supplemented with solar panels. The mass of the system is also less prone to drastic changes; thus, the dynamics of the system are relatively consistent throughout the lifecycle of the spacecraft. Dynamic analyses of small satellites with reaction wheels have been performed in various other graduate projects, such as "Design and Testing of a Nanosatellite Simulator Reaction Wheel Attitude Control System" by Frederic William Long at Utah State University in which a satellite has four reaction wheels arranged in a pyramidal configuration [12]. As the system would be isolated in orbit with no significant external forces acting on it, the torque provided by the motors would impart angular momenta into the system for orientation. This process is described in the following equation.

$$\tau_e = \dot{h}_T = \dot{h}_s + \dot{h}_w \tag{1.1}$$

where h_s is the angular momentum of the system, h_w is the angular moment of the wheel, and τ_e is the total external torque. The angular moment is defined as

$$h = J \times \omega + r \times m \cdot v \tag{1.2}$$

where J is the moment of Inertia, ω is the angular velocity, r is the distance from the center of mass of the system to an arbitrary point about the body, m is the mass of the system, and v is the linear velocity of the system. The torque equation can be further expanded upon so that the angular momentum of the wheels rotates about the body frame as opposed to the wheel frame. This requires a transformation matrix denoted as G to convert the inertia of all four of the wheel frames to body frames. The following equation detail the expansion of the torque equation using Euler's rigid body equation.

$$\tau_e = \dot{h}_s + \omega \times h_s + \dot{h}_w + \omega \times h_w \tag{1.3}$$

If each wheel has the same moment of inertia about its axis of rotation,

$$h_w = J_w \cdot \omega + G \cdot {J_w}^w \cdot \Omega \tag{1.4}$$

Although these equations may be useful in interpreting the full dynamics of the final 3-degree of freedom satellite, outputting an array of the exact angular velocity required may not always be possible. To simplify the dynamics of the system, the primary focus will be on the transfer function of a single motor to platform relationship. Brushed DC motors use permanent magnets and electrical current to spin a shaft, outputting a rated torque which was explained in the above equations [13], but can be simplified to

$$\tau = k_t \cdot I_a \tag{1.5}$$

Where the torque is the product of the given torque constant k_t of the motor and the electrical current I_a flowing through it. While the torque equation and conservation of angular momentum can be used to calculate the angular velocity of the platform, doing so ignores other factors that impede the performance of the DC motor such as mechanical and electrical limitations. The governing equation below is based on Newton's 2nd law and explains the relationship between torque, moment of inertia J and friction b of a DC motor.

$$\tau = J \cdot \dot{\omega} + b \cdot \omega \tag{1.6}$$

All electrical circuits are subject to Kirchhoff's voltage law, which states that the sum of all voltages of all components in the same loop should sum to 0. While this includes the resistance and inductance, it also includes a counter-electromotive force called the back-EMF. This opposes the change in current which induced it and is represented by the following equation.

$$E_b = k_e \cdot \omega \tag{1.7}$$

As the back-EMF is proportional to the angular velocity ω of the shaft spinning in the motor, the main constant that needs to be solved for is the Back EMF constant k_e. In the SI unit, the torque and voltage constants are equal [14].

$$k_t = k_e \tag{1.8}$$

This allows the previously established torque/mechanical equation and the back-EMF equation to be used in Kirchhoff's voltage law.

$$V_a = I_a \cdot R + L_a \cdot I_a + E_b \tag{1.9}$$

These equations will be combined and further expanded upon in a separate section to give the full transfer function for a one-dimensional degree of freedom system as the process of deriving the dynamics of the system and adding additional degrees is simpler than deriving all functions at once. The physical characteristics of the system must also be considered before the full derivation of the transfer function. To house the system, a structure must not only be able to contain, but also accommodate for supplementary material like the electrical system, computational hardware, Guidance, Navigation, and Control, as well as the reactions wheels. This assembly must be able to fit in the payload of a launch vehicle as well, which is why the skeletal structure of this system will be of a large cube shape. The reaction wheel-based CubeSat design is like that of the Cubli self-balancing cube. It operates three reaction wheels along its central axis to balance itself on either its edges or corners [15]. While this design would provide an excellent foundation for the equation of motions, it is different in that the space laser system would rotate about its center of mass as opposed to the edges or corners under gravity. Deriving the Equations of Motion for the structure of the Cube-shaped Space Laser System would require more known values like the hardware characteristics. As a steppingstone, a one-dimensional degree of freedom system would be used instead to derive the equations of motions of the system. Further derivation of this system will be shown in a separate section using the textbook "Dynamics of Mechanical, Aerospace, and Biomechanical Systems Angular Momentum, Inertia, and Angular velocity".

1.3 Outline

The goal of this project is to design and build a satellite capable of detecting certain objects and tracking them before vaporizing them with a laser. As of the time of writing this report, the Department of Defense's global Space Surveillance Network (SSN) estimates that there are more than 27,000 trackable pieces of orbital debris [6]. However, these are the trackable pieces of debris, meaning they are large enough to be picked up by SSN's sensors, which can track objects greater than 10 cm at Low-Earth Orbit. The SSN estimates there may be millions of untrackable orbital debris that could wreak havoc on any spacecraft without the proper protection. Compared to other methods of space debris removal such as Electrodynamic Tethers and Nets, which are more feasible for larger debris [9], lasers seem the most economical for the smaller debris, which will cause the most damage to spacecraft. Post mission analysis of the Space Shuttle launches has estimated that 37% of all impacts on the spacecraft came from paint chips [7], which will be the main target of this satellite.

The reason why this project specifically targets paint is due to the thermodynamic and reflective characteristics of metals over non-metals. This could cause unpredictable reflections that would cause serious consequences, like accidentally hitting a satellite or ablating a material enough to propel it towards another piece of space debris and causing more destruction. As paint chips will come in all shapes and sizes and properties, a uniform object will be used as a stand-in for the paint chip until a method of on-the-fly targeting/learning can be achieved. For this project, a tennis ball will be used as the stand-in for the paint chip. Ultimately, the ideal goal for this project is making target identification as accurate as possible before any destruction which will be addressed in the latter section.

Regarding the construction of the satellite, a Technology Readiness Level of at 6, which is the system/subsystem model or prototype demonstration in a relevant environment [16], is ideal for the scope of this project as higher levels would be prohibitively expensive. Using as many commercially available parts is best for demonstrating the concept while keeping costs and complexity low. Although not strictly a requirement, this project will be in the form of a CubeSat for ease of analytical calculations for the dynamics of the system. This will change depending on the size of the laser. Commercially available laser rust removal tools are well beyond the size of the average CubeSat and may be out of reach without special licensing. As the main objective of this project is tracking, the actual power of the laser is irrelevant, which is why a low-power laser will be used instead. To maximize the lifespan of the project, momentum wheels will be used to orient the satellite in space as opposed to a propulsion system. This will limit the system in that tracking fast-moving objects will be near impossible; however, provided a debris field is detected in advance, an array of satellites can orient themselves ahead of time and activate the lasers as the debris travels through the beams, acting as a net. Depending on the ablation rate of the paint chips, the array of satellites can also sweep across a debris field to cover a much larger area. Further research on the composition of the space-grade paint is needed as these paint chips may not be materially uniform.

Solution methodology

1. Develop a computational framework for object detection: A method for identifying an object from a live video feed is central to this project. There exist commercially available options that still require the user to manually teach the hardware to identify the specific object; however, camera quality may not be ideal. Stronger computation power and machine learning may be required for developing custom tracking software with high-quality cameras.
2. Develop code for object tracking: Orientation and control of the satellite will be performed by momentum wheels. For a singular axis, code will be developed in Arduino IDE with input from the camera to spin the motor about that axis. Depending on the programming language used for the camera, software packages can be used to interact with the Arduino. A PID system would be implemented first via the provided Simulink Autotune feature then be backed by analytical calculations to make movements more precise. An inertial measurement unit will operate as the sensor dynamic to feedback angular velocity data into a closed-loop system. Additional axes will be integrated into the code and Simulink once the singular axis is validated.
3. Create a skeletal structure for the system: The structure will be of a cube shape and will house the electronics suite as well as the motors and the momentum wheels. Once the hardware has been installed into the skeleton, hooks will be installed along the Z and Y axis to test for the balance of the system. Weights will be placed at certain points of the structure to balance the system.
4. Add a laser and design a heat management system: Adding a high-power laser may prove extremely difficult considering the legality of commercially available lasers. As such, a basic low-power laser will be used.
5. Implement additional axes and finalize tuning of PID: As all the necessary hardware has been installed, control over the additional axes can be integrated into the system. The PID and the weighting of variables can be optimized for responsiveness and stability.

The timeline of this project will span from August of 2021 to May of 2022. The object detection aspect is central to this project and software development of this technology will demand the most time. As mentioned before, commercially available options exist; however, the range at which these camera software packages can detect objects is extremely poor. Open-source options for real-time computer vision and will require the use of a micro-computer and additional programming.

Next to the camera and computer selection, the motors are important to the development of this project in that, not only do the motors allow for the project to rotate in space, they also dictate the size of the satellite itself as the frame has to be built around these motors. The motors must be of sufficient torque and speed such that they can spin acrylic momentum wheels fast enough for the satellite to spin in response. Regardless of the choice of programming language for the camera, the method of controlling the motors will be through Arduino IDE due to the number of software packages available for micro-computers to interact with Arduino. The analytical dynamics of a one-dimensional degree of freedom system will be used as the basis for the MATLAB Simulink closed-loop tracking model, which will also include PID tuning and Inertial Measurement Unit feedback. The deadline for developing an analytical code for basic motor tracking functions was November 30. Additional degrees of freedom along the X and Y-Axis may be added when the dynamics about the Z-axis are finalized.

As the hardware has been selected, the skeleton can be constructed around it. Due to the size of high-power motors, the structure will initially be constructed by wood as the system will likely exceed the size of most commercial 3D printers. Once the frame has been completed, the hardware can be installed, and hooks can be installed along the X, Y, and Z-axis of the system. This will allow the system to be freely suspended to test for the balance along an individual axis of the final system. If the satellite is not perfectly level, weights can be added to balance the system. An aluminum skeleton is also being considered, but this method may require special welding not currently available. Further research is required for alternative aluminum structure designs. The construction of the skeleton will coincide with the development of the motor tracking functions along additional axes and will be completed by Spring of 2022. The rest of Spring 2022 was planned to be spent fine-tuning the movement of the satellite with multiple reaction wheels; however, the tracking performance of the controller was not satisfactory enough to warrant expanding to multiple degrees of freedom. Regarding the laser, a basic low-power laser will be used as a stand-in. Commercially available high-power lasers are extremely dangerous and specialized training will be required before obtaining and operating any high-power laser device.

Chapter 2: System Design

Developing the dynamics of the reaction wheel control will require a framework for the dynamic model to be based on. While the orienting the satellite will require multiple reaction wheels, the initial phase of this project will be focused on achieving sufficient tracking response from a singular degree of freedom, particularly the yaw control of the satellite. Once the physical controller for the system has a comparable response to that of the optimized simulated controller, the controller can be replicated for the pitch and roll actuators of the satellite. The dynamics of the system will be based on a physical assembly constructed that will be used in the experimental component of this project, which will only include a single reaction wheel and motor situated on top of a wooden turntable to simulate freedom of movement along the yaw axis.

2.1 Single-Degree of Freedom System

A single degree of freedom system is used to establish the accuracy and responsiveness of the tracking system, particularly the controller. Additional degrees of freedom will be used with the same type of controller for their respective axis. This single degree of freedom system will be based on a physical assembly as recreated in the Solidworks figure below.

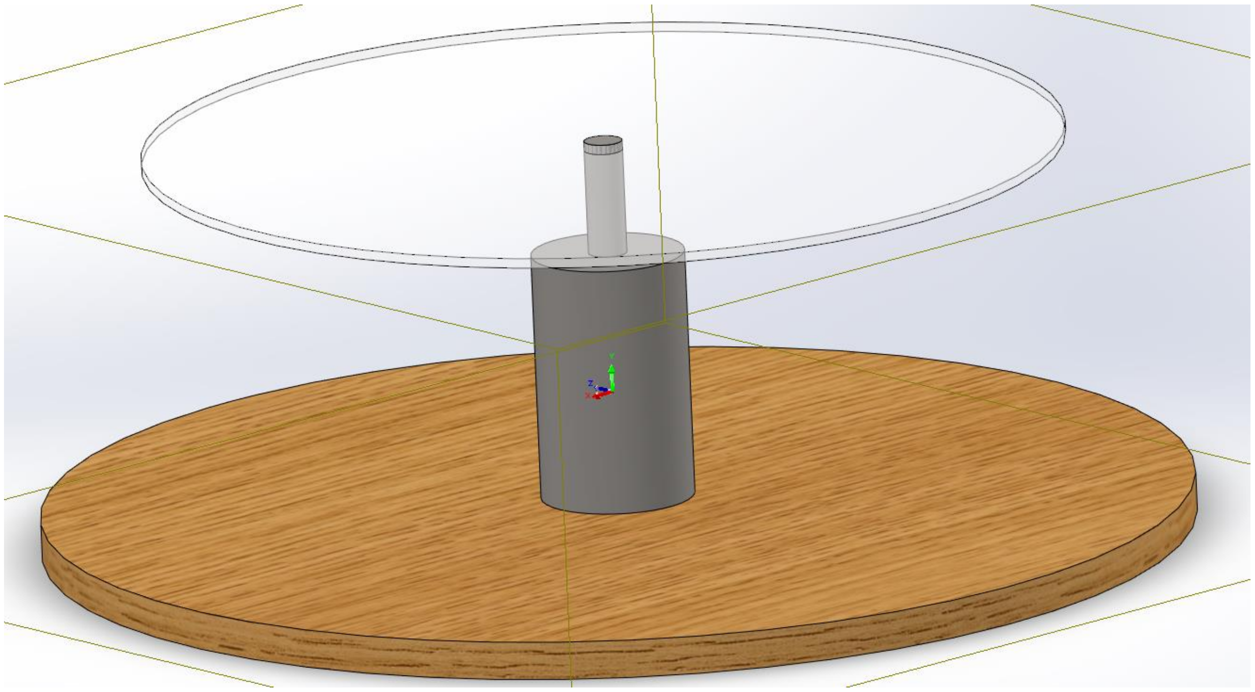


Figure 2.1: Single degree of freedom system model

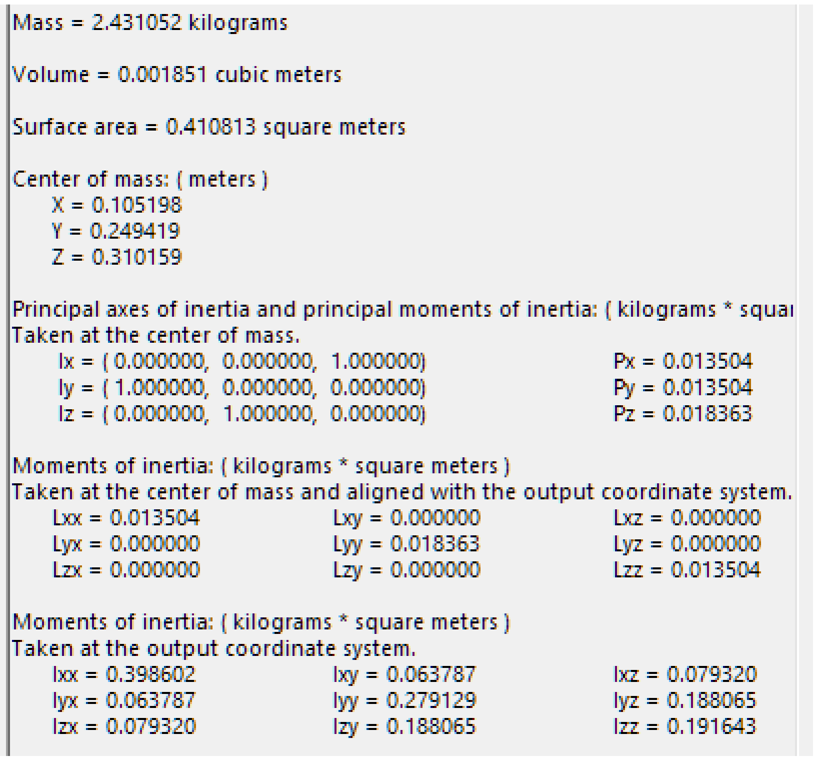

```
Mass = 2.431052 kilograms

Volume = 0.001851 cubic meters

Surface area = 0.410813 square meters

Center of mass: ( meters )
     X = 0.105198
     Y = 0.249419
     Z = 0.310159

Principal axes of inertia and principal moments of inertia: ( kilograms * squa
Taken at the center of mass.
      Ix = ( 0.000000,  0.000000,  1.000000)          Px = 0.013504
      Iy = ( 1.000000,  0.000000,  0.000000)          Py = 0.013504
      Iz = ( 0.000000,  1.000000,  0.000000)          Pz = 0.018363

Moments of inertia: ( kilograms * square meters )
Taken at the center of mass and aligned with the output coordinate system.
     Lxx = 0.013504         Lxy = 0.000000         Lxz = 0.000000
     Lyx = 0.000000         Lyy = 0.018363         Lyz = 0.000000
     Lzx = 0.000000         Lzy = 0.000000         Lzz = 0.013504

Moments of inertia: ( kilograms * square meters )
Taken at the output coordinate system.
     Ixx = 0.398602         Ixy = 0.063787         Ixz = 0.079320
     Iyx = 0.063787         Iyy = 0.279129         Iyz = 0.188065
     Izx = 0.079320         Izy = 0.188065         Izz = 0.191643
```

Figure 2.2: Moment of inertia of single degree of freedom system

The wooden platform is situated on top of a fixed ball-bearing stand so that the wooden platform can freely rotate along the axis normal to the face of the platform. The platform will also serve as the base for the electronics suite that will interface with the 12V DC brushed motor located at the center of the wooden platform. The system will orient itself by spinning an acrylic reaction wheel affixed to the motor shaft, applying a torque in the opposite direction due to conservation of angular momentum. The moment of inertia of the reaction wheel and the motor is given by the following Solidworks figure.

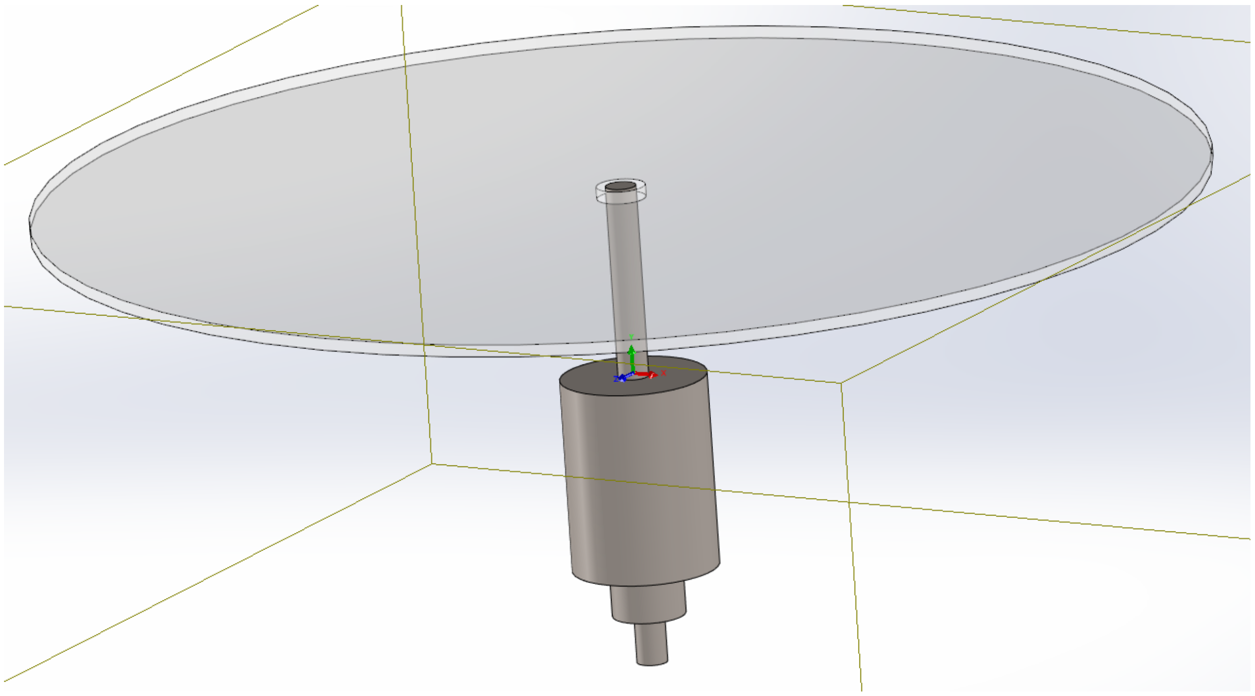
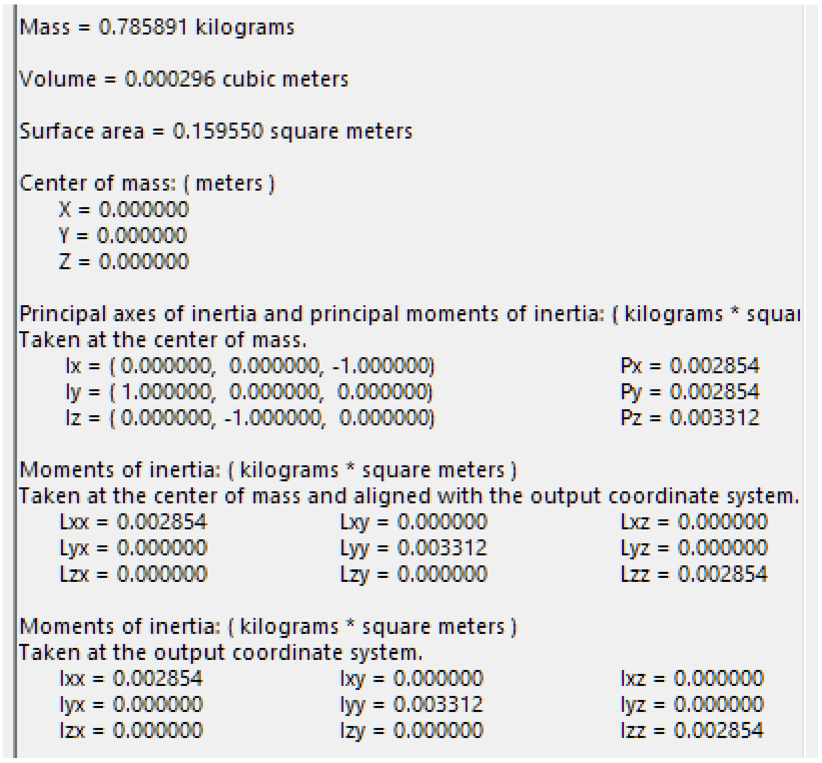


Figure 2.3: Model of motor shaft and reaction wheel



```
Mass = 0.785891 kilograms

Volume = 0.000296 cubic meters

Surface area = 0.159550 square meters

Center of mass: ( meters )
     X = 0.000000
     Y = 0.000000
     Z = 0.000000

Principal axes of inertia and principal moments of inertia: ( kilograms * squa
Taken at the center of mass.
      Ix = ( 0.000000,  0.000000, -1.000000)             Px = 0.002854
      Iy = ( 1.000000,  0.000000,  0.000000)             Py = 0.002854
      Iz = ( 0.000000, -1.000000,  0.000000)             Pz = 0.003312

Moments of inertia: ( kilograms * square meters )
Taken at the center of mass and aligned with the output coordinate system.
     Lxx = 0.002854            Lxy = 0.000000            Lxz = 0.000000
     Lyx = 0.000000            Lyy = 0.003312            Lyz = 0.000000
     Lzx = 0.000000            Lzy = 0.000000            Lzz = 0.002854

Moments of inertia: ( kilograms * square meters )
Taken at the output coordinate system.
     Ixx = 0.002854            Ixy = 0.000000            Ixz = 0.000000
     Iyx = 0.000000            Iyy = 0.003312            Iyz = 0.000000
     Izx = 0.000000            Izy = 0.000000            Izz = 0.002854
```

Figure 2.4: Moment of inertia of motor shaft and reaction wheel

2.1.1 Platform

The system is situated on a wooden turntable where a large platform is attached to a fixed base via a rotary bearing, allowing the platform to freely rotate about the axis normal to its face. This platform measures 15 inches in diameter and 0.75 inches in thickness and weighs 1650 grams.

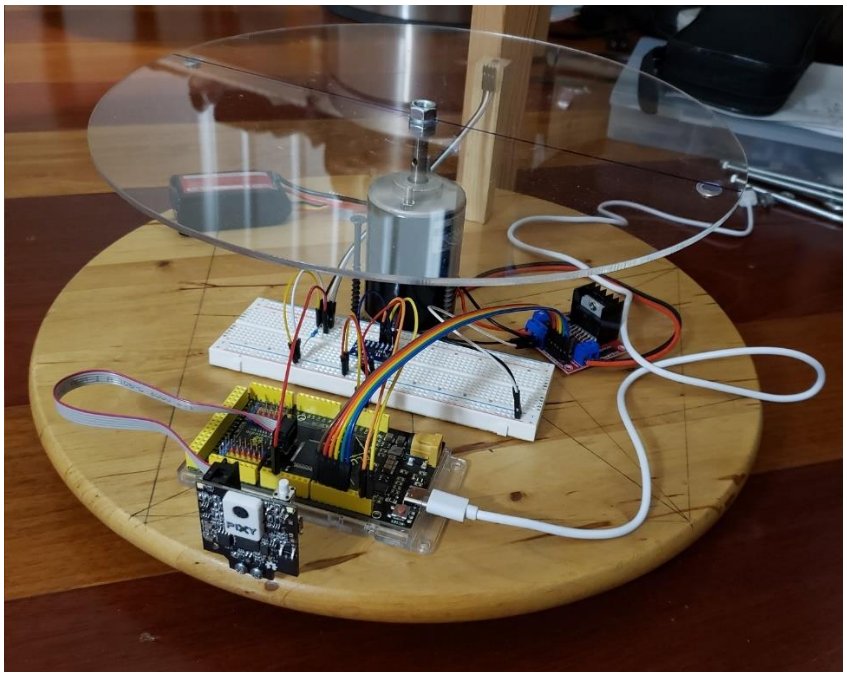


Figure 2.5: Single degree of freedom experimental setup

An alternate design that was considered is a round wooden plate suspended via a swivel lock as shown below.

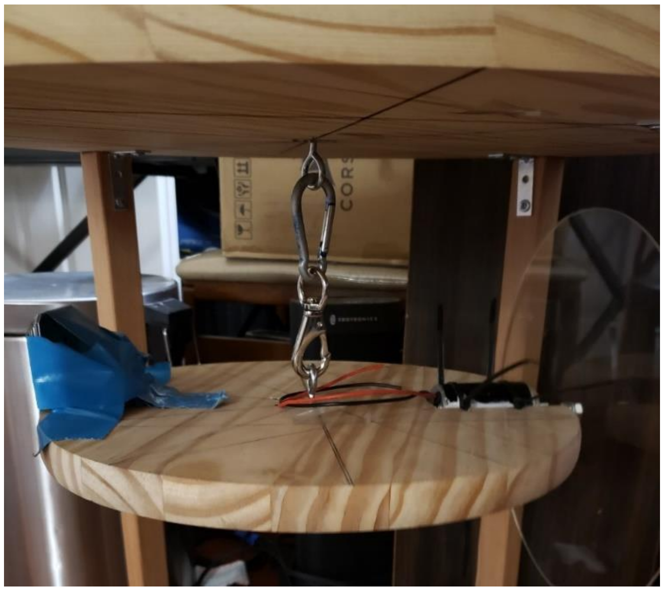


Figure 2.6: Alternate platform design

Due to its suspended design, the system will be elevated above the ground which allows a motor and reaction wheel to be placed on the underside of the platform. This design would also allow for additional motors and reaction wheels to be placed along the pitch and roll axis, enabling three degrees of freedom for the system. While this design was initially planned as a base model for the project moving forward, it is more susceptible to perturbations that could affect the results of testing. As the focus of this project is to optimize the tracking performance of a satellite, the ball-bearing turntable will be used as a base model for designing the actuator controllers. Once the controllers have comparable performance to that of the simulation, the system will transition to this design for multiple degrees of freedom.

2.1.2 Reaction Wheel

While there are numerous methods for adjusting the orientation of satellites, reactions wheels were chosen for this system for its low operational costs and long lifespan due to relying solely on electricity to generate enough angular momentum to rotate the body. The reaction wheel used in this system is a 12-inch acrylic disc and weighs 240 grams. The disc was purchased from a store that specializes in custom polymer pieces, in which they drilled an eight-millimeter hole in the center of the disc, which allows it to be held onto the threaded motor shaft with nuts. Alternate reaction wheels designs such as aluminum ring discs have been considered; however, a larger reaction wheel made of metal would require custom machining that is not immediately available. When the system is redesigned for spaceflight, metal ring discs would be preferable due to their larger moment of inertia.

2.1.3 Motor and Motor Driver

While the conservation of angular momentum can be used to explain the momentum exchange between the reaction wheel and the platform, obtaining the correct output speed of the reaction wheel requires the modeling of the actuator dynamics between the input voltage and the output angular velocity of the motor shaft reaction wheel. The motor used in the single degree of freedom system is the XD-3420 brushed DC motor with the following specifications [17]:

- Rated Voltage: 12V
- No-Load Speed: 3500 RPM
- Rated Current: 400 mA
- Rated Torque: 0.196 N-m
- Inductance: 0.001339 Henry
- Resistance: 2.3 ohm
- Viscous Friction Coefficient: 0.000002

While the motor can provide the system with the angular momentum required to turn the platform, this system will require precise control over the motor speed and direction. Most commercially available computers cannot directly interface with these motors due to their large power requirements; this can be mitigated by using the L298N H-bridge motor driver. By using an external power source such as high voltage batteries, the motor driver will interpret the received signal and scale with the power supply to power the motor. The L298N module is a generic motor driver that can operate a 12V load as well as power the Arduino unit. Below are the specifications for this motor driver [18]:

- Motor Voltage: 3.2-40V
- Supply Driver Voltage: 5-35V
- Supply Driver Current:2A
- Peak Current: 2A
- Logic Voltage: 5V
- Logical Current:0-36mA
- Maximum Power (W): 25W

Despite being able to operate the XD3420 motor, one issue that could pose a threat to the lifecycle of the satellite was the dangerous amounts of heat being produced from the motor driver. A temperature reading of close to 100 degrees Fahrenheit was taken off the L298N motor driver during operation.

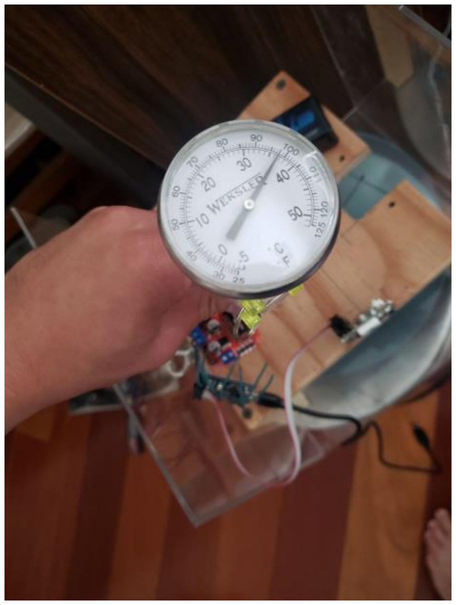


Figure 2.7: Temperature reading of the L298N motor driver

Alternate motor drivers are being considered, such as the TB6612FNG, which operate on modern MOSFETs [19]. Below are the specifications of the TB6612FNG motor driver:

- Motor Voltage: 15V
- Supply Driver Voltage: 6V
- Peak Current: 2A
- Logic Voltage: 5.5V
- Logical Current: 0.4A

While comparing the specifications of the two motor drivers directly, the L29N seems superior; however, the TB6612FNG motor driver has much less power dissipation which should produce much less heat. However, as the focus of this project is on the control dynamics of the satellite, the motor driver that will be used for the satellite will be the L298N, as implementing the

TB6612FNG will require more software development that offer little benefits to the tracking performance of the system outside of longer test sample sizes.

2.1.4 Transfer Function

With the physical characteristics of the platform, wheel, and motor defined, the transfer function or actuator dynamics between the platform and the voltage sent from microcontroller can be modelled. The actuator dynamics of the system describe the process of converting signal, or a certain amount of voltage, into physical motion through a DC motor. As the set-up is a one degree of freedom system, there is only one transfer function in which current applied to the motor would output an angular velocity onto the platform. This transfer function can be obtained by the following equations established in the literature review.

$$\tau = k_t \cdot I_a \tag{2.1}$$

$$\tau = J \cdot \dot{\omega} + b \cdot \omega \tag{2.2}$$

$$E_b = k_e \cdot \omega \tag{2.3}$$

$$V_a = I_a \cdot R + L_a \cdot \dot{I}_a + E_b \tag{2.4}$$

The first step in obtaining the transfer function is to take the Laplace Transform of these four equations. This allows complex differential equations to be solvable in the frequency domain or s-domain which removes the time dependency from these functions.

$$\tau(s) = k_t \cdot I_a(s) \tag{2.5}$$

$$\tau(s) = J \cdot s \cdot \omega(s) + b \cdot \omega(s) \tag{2.6}$$

$$E_b(s) = k_e \cdot \omega(s) \tag{2.7}$$

$$V_a(s) = I_a(s) \cdot R + L_a \cdot s \cdot I_a(s) + E_b(s) \tag{2.8}$$

As the time-domain is no longer a factor, the torque equations (3.5 & 3.6) can be combined, and the current can be isolated.

$$k_t \cdot I_a(s) = J \cdot s \cdot \omega(s) + b \cdot \omega(s)$$

$$I_a = \frac{J \cdot s \cdot \omega(s) + b \cdot \omega(s)}{k_t} \tag{2.9}$$

The current equation (3.9) as well as the back-EMF equation (3.7) can be used in the Kirchhoff's Voltage Law (3.8) as shown below.

$$V_a(s) = \frac{J \cdot s \cdot \omega(s) + b \cdot \omega(s)}{k_t} \cdot R + L_a \cdot s \cdot \frac{J \cdot s \cdot \omega(s) + b \cdot \omega(s)}{k_t} + k_e \cdot \omega(s)$$

$$V_a(s) = \frac{J \cdot s + b}{k_t} \cdot \omega(s) \cdot (R + L_a \cdot s) + k_e \cdot \omega(s)$$

$$V_a(s) = \frac{(J \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t}{k_t} \cdot \omega(s) \tag{2.10}$$

As stated in the literature review, $k_e = k_t$. As such, the transfer function from the input voltage into a motor to the angular velocity outputted by the reaction wheel is given by the following transfer function.

$$\frac{\omega(s)}{V_a(s)} = \frac{k_t}{(J \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t} \tag{2.11}$$

This transfer function is only concerning the relationship between the voltage input and the angular velocity of the reaction wheel; it does not output the angular velocity of the platform. The relationship between the angular velocity of the reaction wheel and the platform can be explained by the conservation of angular momentum.

$$J_{wheel} \times \omega_{wheel} = J_{system} \times \omega_{system}$$

$$\omega_{system} = \frac{J_{wheel}}{J_{system}} \times \omega_{wheel} \tag{2.12}$$

The angular velocity of the satellite is given by the following transfer function

$$\frac{\omega_{system}(s)}{V_a(s)} = \frac{J_{wheel}}{J_{system}} \times \frac{k_t}{(J_{wheel} \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t} \tag{2.13}$$

This transfer function will be used as the basis for modeling the simulation in MATLAB Simulink. The system will track objects by minimizing the angular displacement between the location of tracked object and the viewing axis of the camera system. The format of the transfer function in equation (2.13) is only able to output the angular velocity but can be modified to output angular displacement by replacing it with the derivative of the angular displacement. Since the transfer function is already in the s-domain, the Laplace transform of the derivative of the angular displacement can replace the angular velocity in the transfer function. The s variable can then be transferred to the right-hand side, which increases the rank of the transfer function from two to three, meaning that the transfer function will output the signal current, angular velocity and angular displacement.

$$\omega_{system} = \frac{d}{dt}\theta \tag{2.14}$$

$$\omega_{system}(s) = \theta(s) \cdot s$$

$$\frac{\theta(s) \cdot s}{V_a(s)} = \frac{J_{wheel}}{J_{system}} \times \frac{k_t}{(J_{wheel} \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t}$$

$$\frac{\theta(s)}{V_a(s)} = \frac{J_{wheel}}{J_{system}} \times \frac{k_t}{((J_{wheel} \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t) \cdot s} \tag{2.15}$$

2.2 Two-Degree of Freedom System

Gravity-gradient stabilization is defined as the alignment of one axis of a satellite along the earth's local vertical direction so that the end of the satellite aligned with that axis will always face down relative to the surface of the earth. There are two notable benefits associated with this; one of which is that this configuration allows for enhanced signal strength for communication devices, minimizing the potential lag that may arise when tracking debris. The other is that this configuration minimizes reaction wheel usage for stabilization, thereby minimizing energy usage [20]. Gravity-gradient stabilization allows for the system to be controlled by only two reaction wheels; however, the satellite must be constructed in a manner that allows for this. To achieve gravity-gradient stabilization, the inertia of the roll, pitch, and yaw must be influenced to create the following yaw and roll constants.

$$K_Y = \frac{I_{pitch} - I_{roll}}{I_{yaw}} \tag{2.16}$$

$$K_R = \frac{I_{pitch} - I_{yaw}}{I_{roll}} \tag{2.17}$$

To achieve gravity-gradient stabilization, the system must satisfy the following conditions

$$K_Y < K_R \tag{2.18}$$

$$3 \cdot K_R + K_Y \cdot K_R + 1 > 4 \cdot \sqrt{K_Y \cdot K_R} \tag{2.19}$$

$$K_Y \cdot K_R > 0 \tag{2.20}$$

Provided that these conditions were met, this would allow the satellite to be controlled by two motors centered on the yaw and roll axis. Using the two degree of freedom system that was been previously designed as a base model, the system has been constructed and recreated in Solidworks. To ensure the system is balanced along the axis normal to the center of the face of the platform, a counterweight it placed opposite to the pitch axis motor.

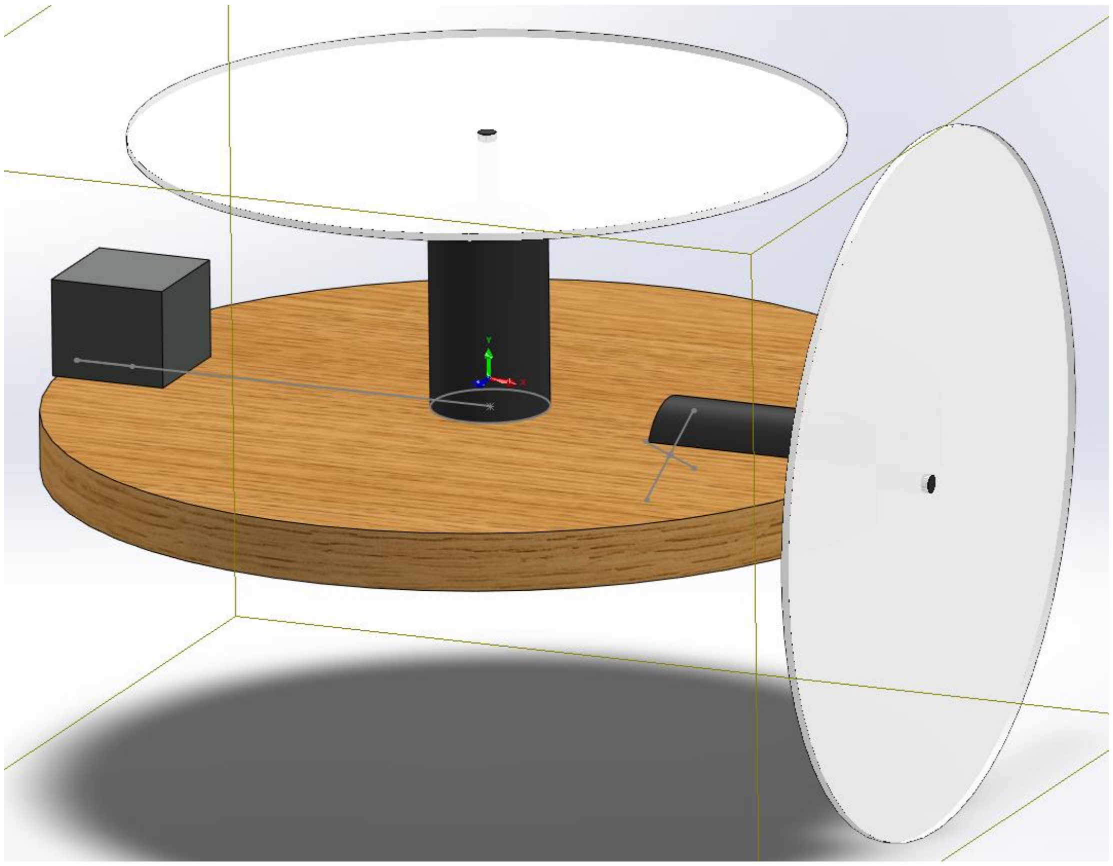
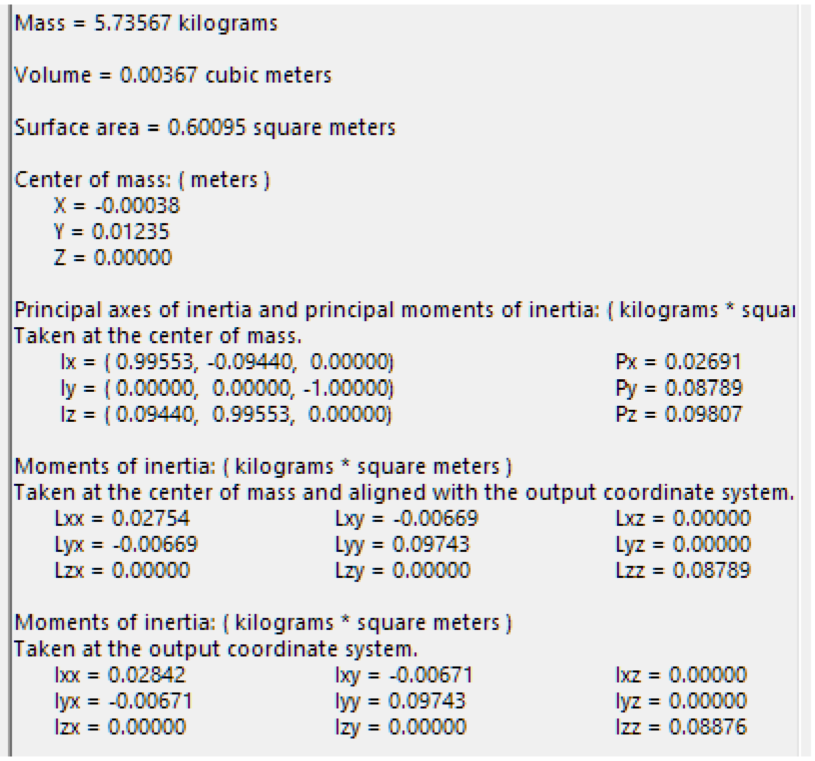


Figure 2.8: Two degree of freedom system model



```
Mass = 5.73567 kilograms

Volume = 0.00367 cubic meters

Surface area = 0.60095 square meters

Center of mass: ( meters )
     X = -0.00038
     Y = 0.01235
     Z = 0.00000

Principal axes of inertia and principal moments of inertia: ( kilograms * squa
Taken at the center of mass.
      Ix = ( 0.99553, -0.09440,  0.00000)                 Px = 0.02691
      Iy = ( 0.00000,  0.00000, -1.00000)                 Py = 0.08789
      Iz = ( 0.09440,  0.99553,  0.00000)                 Pz = 0.09807

Moments of inertia: ( kilograms * square meters )
Taken at the center of mass and aligned with the output coordinate system.
     Lxx = 0.02754             Lxy = -0.00669             Lxz = 0.00000
     Lyx = -0.00669            Lyy = 0.09743              Lyz = 0.00000
     Lzx = 0.00000             Lzy = 0.00000              Lzz = 0.08789

Moments of inertia: ( kilograms * square meters )
Taken at the output coordinate system.
     Ixx = 0.02842             Ixy = -0.00671             Ixz = 0.00000
     Iyx = -0.00671            Iyy = 0.09743              Iyz = 0.00000
     Izx = 0.00000             Izy = 0.00000              Izz = 0.08876
```

Figure 2.9: Moment of inertia of two degree of freedom system

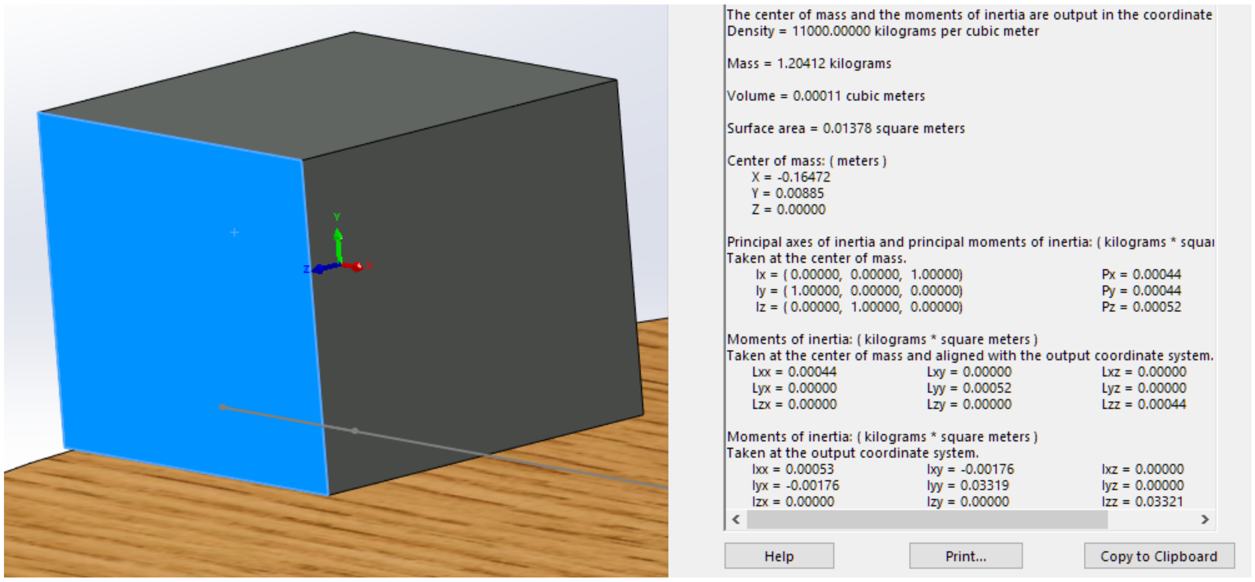

```
The center of mass and the moments of inertia are output in the coordinate
Density = 11000.00000 kilograms per cubic meter

Mass = 1.20412 kilograms

Volume = 0.00011 cubic meters

Surface area = 0.01378 square meters

Center of mass: ( meters )
     X = -0.16472
     Y = 0.00885
     Z = 0.00000

Principal axes of inertia and principal moments of inertia: ( kilograms * squa
Taken at the center of mass.
      Ix = ( 0.00000,  0.00000,  1.00000)                    Px = 0.00044
      Iy = ( 1.00000,  0.00000,  0.00000)                    Py = 0.00044
      Iz = ( 0.00000,  1.00000,  0.00000)                    Pz = 0.00052

Moments of inertia: ( kilograms * square meters )
Taken at the center of mass and aligned with the output coordinate system.
     Lxx = 0.00044              Lxy = 0.00000              Lxz = 0.00000
     Lyx = 0.00000              Lyy = 0.00052              Lyz = 0.00000
     Lzx = 0.00000              Lzy = 0.00000              Lzz = 0.00044

Moments of inertia: ( kilograms * square meters )
Taken at the output coordinate system.
     Ixx = 0.00053              Ixy = -0.00176             Ixz = 0.00000
     Iyx = -0.00176             Iyy = 0.03319              Iyz = 0.00000
     Izx = 0.00000              Izy = 0.00000              Izz = 0.03321
```

Figure 2.10: Counterweight for two degree of freedom system

Given the moment of inertias of the yaw and roll of the system, this two degree of freedom system design would not satisfy the first and second conditions for gravity gradient stabilization. To satisfy the requirements for gravity gradient stabilization, the system would need to be modified to increase the moment of inertia of the roll axis by adding mass to the center of the face of the platform as well as elongating the material to increase the height of the system. Due to the elevated design, testing of this system would require it to be suspended to increased freedom of movement at the cost of stability that could interfere with initial testing. As previously mentioned, the purpose of this project is to optimize the tracking performance of the system and the single degree of freedom system will be used as the base model to optimize tracking in the yaw direction before including additional degrees of freedom.

Chapter 3: System Modeling and Controller Design

The tracking performance of the system has two parallel process that will be used to validate each other. The experimental process will consist of testing and analyzing the physical setup established in the framework layout, while the computational process will recreate the physical setup and the test conditions. The results of these processes will be compared to each other to validate the accuracy of the simulation, which can then be used to model multiple degrees of freedom.

3.1 Open-Loop Stability Analysis

The computational process involves recreating the single degree of freedom system in MATLAB Simulink, which would require the transfer function previously established in equation (2.13) to accurately model the motor in the hardware setup. To simulate the transfer function, it must be developed into state space form, which allows complex transfer functions to be reinterpreted as a system of first order differential equations. This can be achieved by reorganizing the transfer function in the following equations but refer to appendix A for full derivation.

$$\frac{\omega_{system}(s)}{V_a(s)} = \frac{J_{wheel} \cdot k_t}{J_{system} \cdot (J_{wheel} \cdot s \cdot R + L_a \cdot s^2 \cdot J_{wheel} + b \cdot R + L_a \cdot s \cdot b) + J_{system} \cdot k_e \cdot k_t} \tag{3.1}$$

$$\frac{\omega_{system}(s)}{V_a(s)} = \frac{J_{wheel} \cdot k_t}{(J_{system} \cdot J_{wheel} \cdot L_a) \cdot s^2 + ((J_{wheel} \cdot R + L_a \cdot b) \cdot J_{system}) \cdot s + (b \cdot R + k_e \cdot k_t) \cdot J_{system}} \tag{3.2}$$

$$\frac{\theta(s) \cdot s}{V_a(s)} = \frac{J_{wheel} \cdot k_t}{(J_{system} \cdot J_{wheel} \cdot L_a) \cdot s^2 + ((J_{wheel} \cdot R + L_a \cdot b) \cdot J_{system}) \cdot s + (b \cdot R + k_e \cdot k_t) \cdot J_{system}} \tag{3.3}$$

$$\frac{\theta(s)}{V_a(s)} = \frac{J_{wheel} \cdot k_t}{(J_{system} \cdot J_{wheel} \cdot L_a) \cdot s^3 + ((J_{wheel} \cdot R + L_a \cdot b) \cdot J_{system}) \cdot s^2 + (b \cdot R + k_e \cdot k_t) \cdot J_{system} \cdot s} \tag{3.4}$$

Obtaining the state space of the transfer function is done by using the function tf2ss which requires the coefficients of each order of the s-domain in both the numerator and the denominator to be inputted into the b and a value of the tf2ss function respectively. From the function, the outputs are the A, B, C, and D matrices which forms the basis for the state space system that can be replicated in MATLAB Simulink. These values are

$$A = 10^4 \begin{bmatrix} -0.1718 & -5.4174 & 0 \\ 0.0001 & 0 & 0 \\ 0 & 0.0001 & 0 \end{bmatrix} \tag{3.5}$$

$$B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.6}$$

$$C = 10^4[0 \quad 0 \quad 1.9932] \tag{3.7}$$

$$D = [0] \tag{3.8}$$

The state space form can be defined a

$$\frac{d}{dt}\begin{bmatrix} i \\ \dot{\theta} \\ \theta \end{bmatrix} = 10^4 \begin{bmatrix} -0.1718 & -5.4174 & 0 \\ 0.0001 & 0 & 0 \\ 0 & 0.0001 & 0 \end{bmatrix} \begin{bmatrix} i \\ \dot{\theta} \\ \theta \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} V \tag{3.9}$$

$$y = 10^4[0 \quad 0 \quad 1.9932]\begin{bmatrix} i \\ \dot{\theta} \\ \theta \end{bmatrix} \tag{3.10}$$

To ensure that the actuator of the system can control all states of the system, it must be evaluated for controllability. This is achieved by checking if the controllability matrix is a full rank matrix. The controllability matrix is obtained by

$$\mathrm{Co} = [B \quad AB \quad A^2B] \tag{3.11}$$

$$\mathrm{Co} = \begin{bmatrix} 1 & -1717 & 2896320 \\ 0 & 1 & -1717 \\ 0 & 0 & 1 \end{bmatrix}$$

$$det(Co) = 10^4 \begin{vmatrix} 1 & -1717 & 2896320 \\ 0 & 1 & -1717 \\ 0 & 0 & 1 \end{vmatrix} = 1 \tag{3.12}$$

The controllability matrix can be determined to be full rank by taking the derivative of the controllability matrix. If the determinant of the controllability of the matrix is a non-zero value, then the system is full rank, and thus fully controllable. Taking the determinant of the controllability matrix, the determinant is equal to 1, thus the system is fully controllable.

Despite being controllable, the simulated system may not be able to accurately output the known status of the certain variables. As such, the observability of the system must be determined. This is achieved in the same manner as the controllability matrix in that the observability matrix must be full rank or linearly independent. The observability matrix is obtained by

$$\mathrm{Ob} = \begin{bmatrix} C \\ CA \\ CA^2 \end{bmatrix} \tag{3.13}$$

$$\mathrm{Ob} = 10^4 \begin{bmatrix} 0 & 0 & 1.9932 \\ 0 & 1.9932 & 0 \\ 1.9932 & 0 & 0 \end{bmatrix}$$

$$det(Ob) = 10^4 \begin{vmatrix} 0 & 0 & 1.9932 \\ 0 & 1.9932 & 0 \\ 1.9932 & 0 & 0 \end{vmatrix} = -7.9182 \cdot 10^{12} \qquad (3.14)$$

The determinant of the observability matrix is a non-zero value; thus, the system is fully observable.

While the system is fully observable, the system's angular displacement from its initial position is unknown by the system as there is no feedback loop to validate its current position with respect to the reference frame, thus making this an open-loop system. Provided the system is programmed to rotate itself to an object 10 degrees away, without a feedback loop to validate its current angular displacement, the system will continuously rotate as the 10-degree reference sign will be continuously moving. Below is model and plot of the open-loop system.

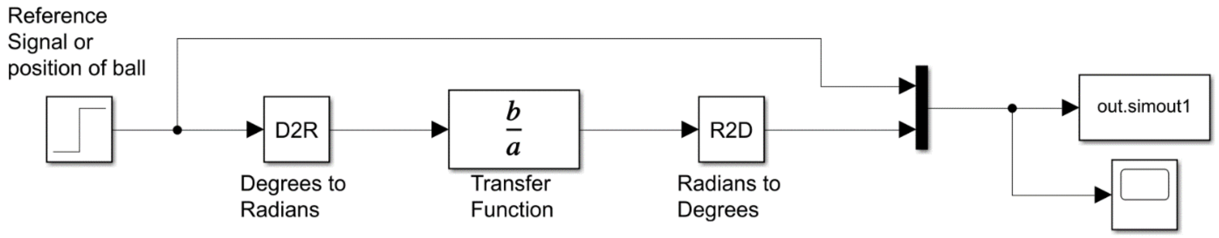


Figure 3.1: Open-loop Simulink diagram

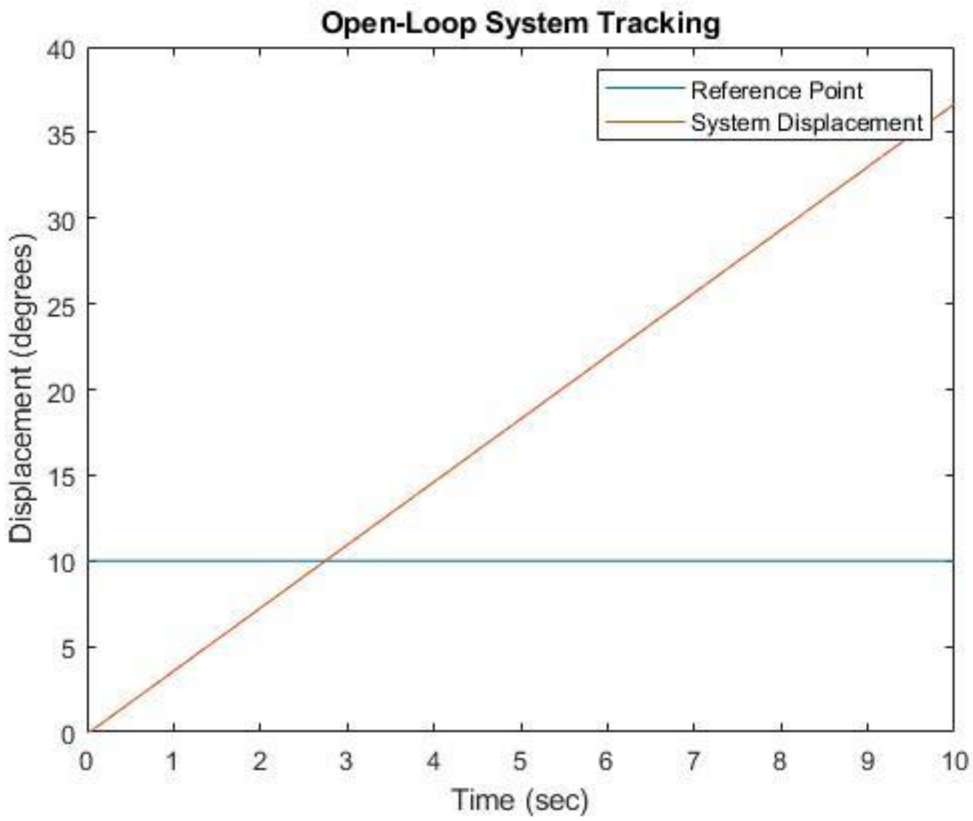


Figure 3.2: Open-loop system response

3.2 Closed-Loop Stability Analysis

A feedback loop of the output to the reference signal allows the system to register its angular displacement for improved tracking performance and allow it to be reclassified as a closed-loop system. While the feedback loop will allow the system to track its angular displacement relative to the object, it does so at an incredibly slow rate. Using the reference signal and a time of 10 seconds, the system was not able to reach the reference signal within the given timeframe. A proportional gain can be used to increase the speed at which the system approaches the tracked object; however, it will still use the same speed for short distances as well, which would cause the system to overshoot. To accurately control the system in a timely manner, a PID controller will be placed in the system. To reach the reference angle as efficiently as possible a PID controller into the system.

Proportional-Integral-Derivative (PID) controllers are control loop mechanisms used to regulate the output of a system to match the desired reference value. The basic form of the PID system is given by the following equation

$$Output = K_p \cdot error(t) + K_i \cdot \int error(t) \cdot dt + K_d \cdot \frac{d}{dt} error(t) \tag{3.15}$$

The values of the K_p, K_i, K_d are to be chosen by the user based of their performance preferences. The K_p will provide a fixed rate at which the system with rotate. The K_d increase the speed of the system based on how much further the reference signal from its current position, thus increasing the responsiveness of the system. In the case of the single degree of freedom system, a high K_d causes the system to overshoot. The K_i value measures the change in error of the system and slows down the input, steadying the system. Below is system model with PID controller.

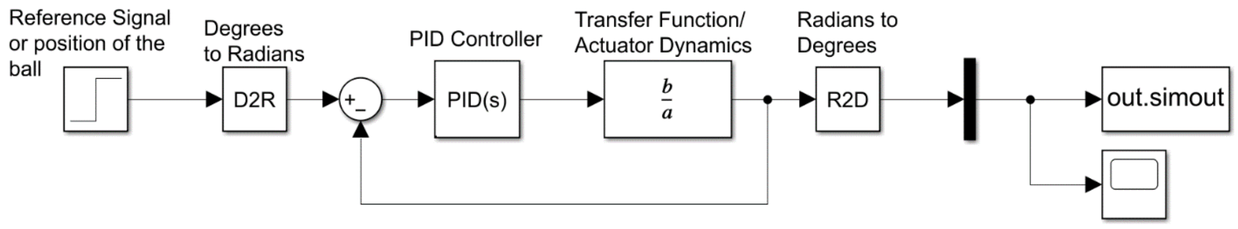


Figure 3.3: Closed-Loop PID controller Simulink diagram

MATLAB Simulink includes a PID block which an autotune feature that allows users to adjust the PID values based on the user's preference for response time and transient behavior. Tuning the system for maximum possible response time and aggressive behavior provides the fastest possible response time, but introduces large oscillations in the system as shown in the diagram below

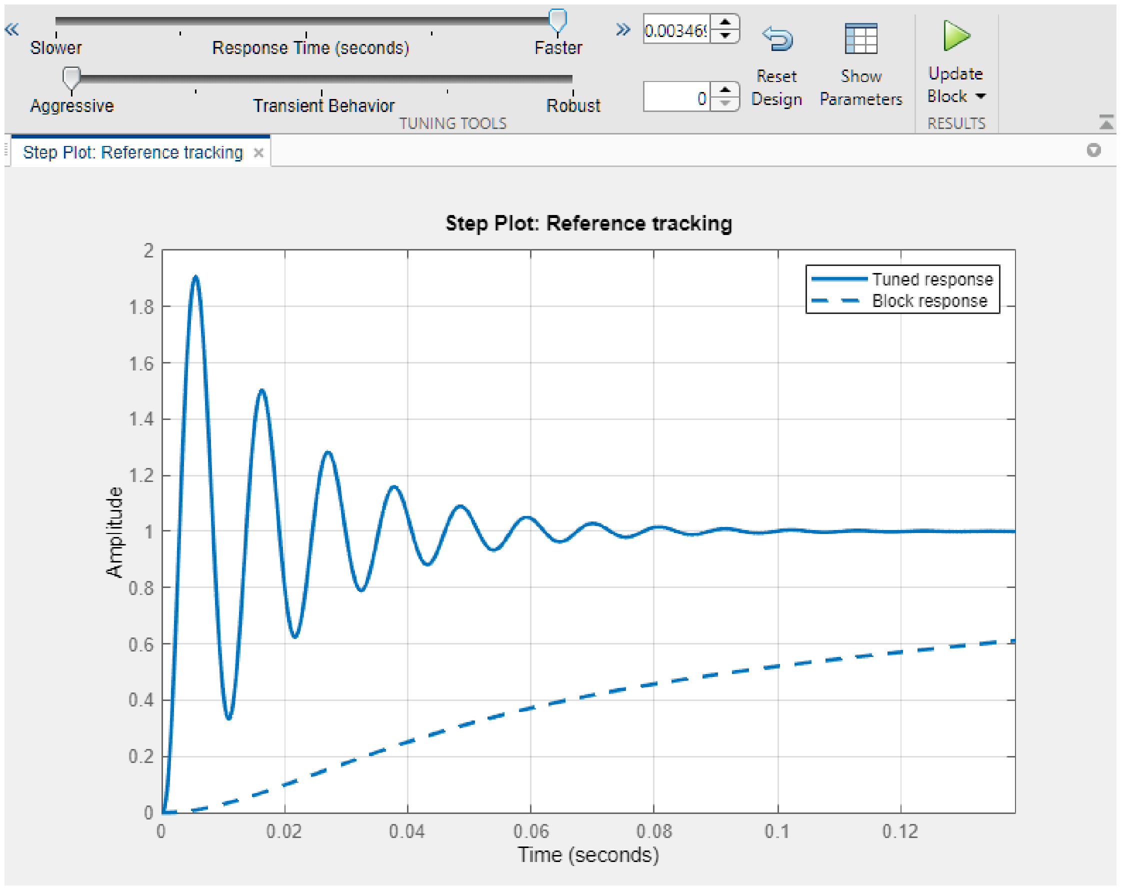
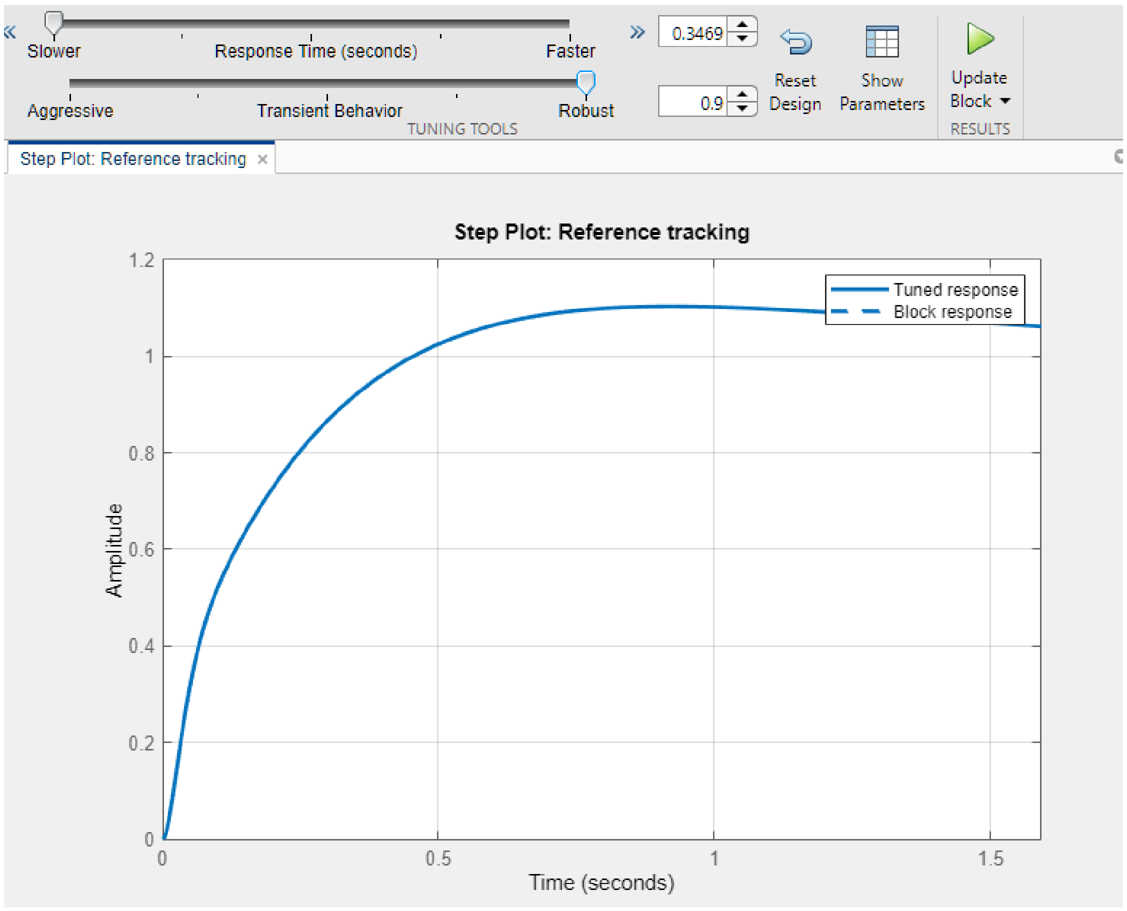


Figure 3.4: PID autotune for maximum speed and aggression

The PID values chosen for this project emphasizes slow reaction speed and robust behavior. While not as fast as the previous option, these PID values provide sufficient speed for tracking objects at roughly half a second as shown below.



Response Time (seconds): 0.3469
Transient Behavior: 0.9

Figure 3.5: PID autotune for slow speed and stability

The PID values from this configuration are listed below and provides the following tracking performance

$$K_p = 15.142$$

$$K_i = 14.922$$

$$K_d = 0.954$$

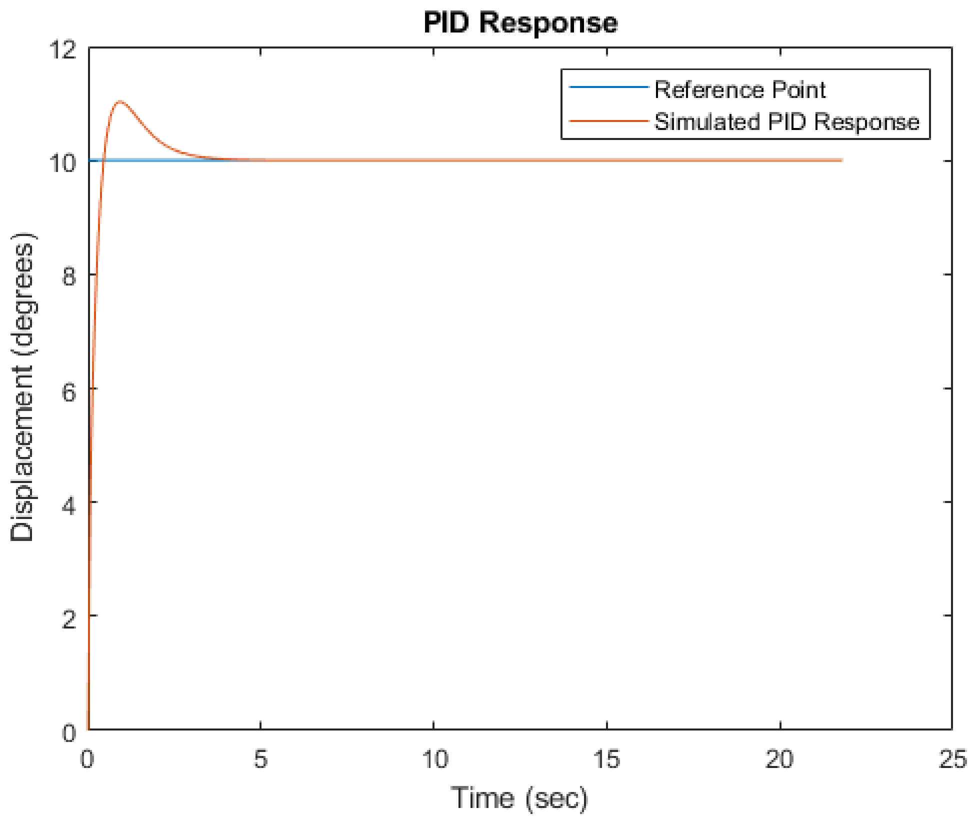


Figure 3.6: Simulated PID response vs reference signal

The response of the system, while sufficiently fast, overshoots the reference point, which can cause issues with the code as the code operates of "if greater than or less than" statements that could cause the system to jerk violently. This can be mitigated through use of buffers or deadzones, but this issue could still occur. Adjusting the PID values is still an option but designing a more accurate controller may be preferable.

The Linear Quadratic Regulator (LQR) is a modern control theory method based optimizing performance and cost based on the user's preferences, often by adjusting the Q and R matrices which acts as the weighting of the system. Modeling the LQR controller requires the state space form of the transfer function previously derived in equation (3.9) and (3.10). During the initial phases of designing the PID and LQR controllers, the transfer function of the system was incorrectly modeled based on the transfer function derived in equation (2.13). While the equation is correct, it does not give the correct output as the output of the function is the angular velocity of the system. This mean that at a reference value of 10, the output of the system would be 10 degrees per second. This in turn would also output incorrect PID and state space controller values. While this issue can be resolved for the PID controller by placing an integrator immediately after the transfer function, this method does not work for the LQR controller. Below is the model of the system using the incorrect transfer function equation (2.13) with an integrator as well as the system response.

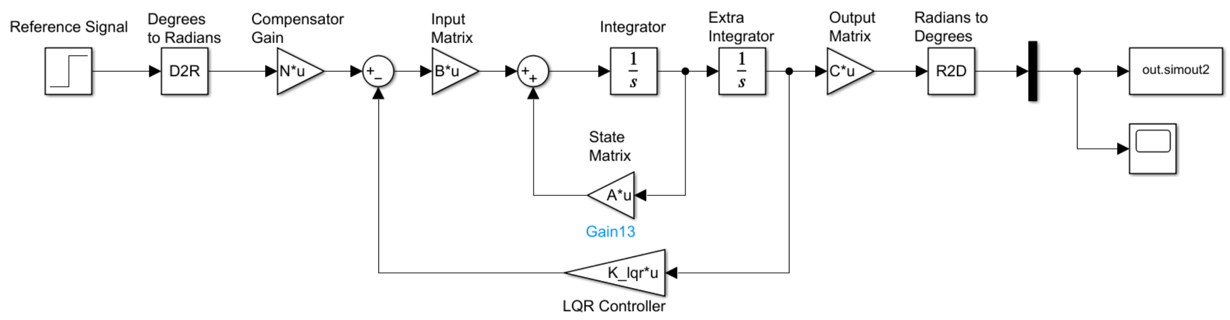


Figure 3.7: Initial attempt at LQR controller Simulink diagram

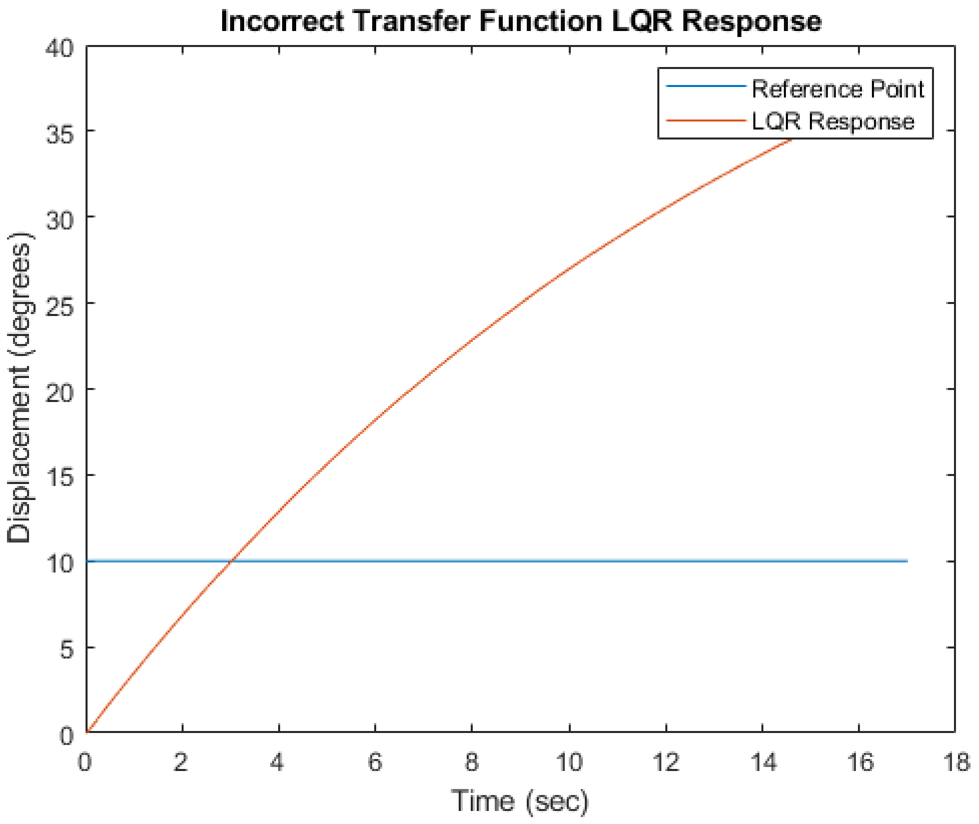


Figure 3.8: Initial attempt LQR response

Any adjustment of the Q and R matrices have no impact on the tracking performance of the system and further development could not be made with this model of the system. This led to the reevaluation of the transfer function of the system into what was developed in equation (3.4).

To retrofit the Simulink model for the transfer function in equation (3.4), the second integrator before the K loop can be deleted and the input into the tf2ss function can be changed to match the numerators and denominators of equation (3.4). The Q and R matrices should also be defined, starting as identity matrix to be configured by the user to meet the design requirements. An N gain value is placed before the LQR controller to compensate for the weak reference signal. Below is system model with LQR controller recreated in MATLAB Simulink as well as the LQR controller's response at Q being equal to the identity matrix and R being equal to 1.

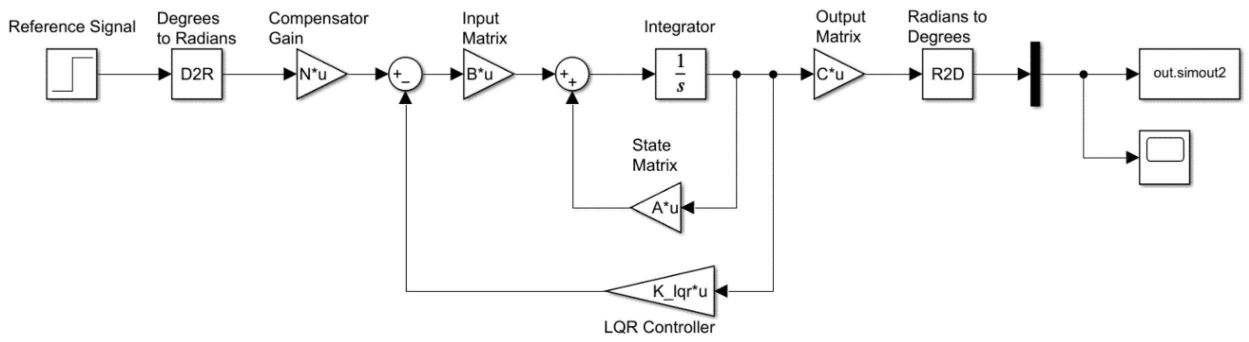


Figure 3.9: LQR controller Simulink diagram

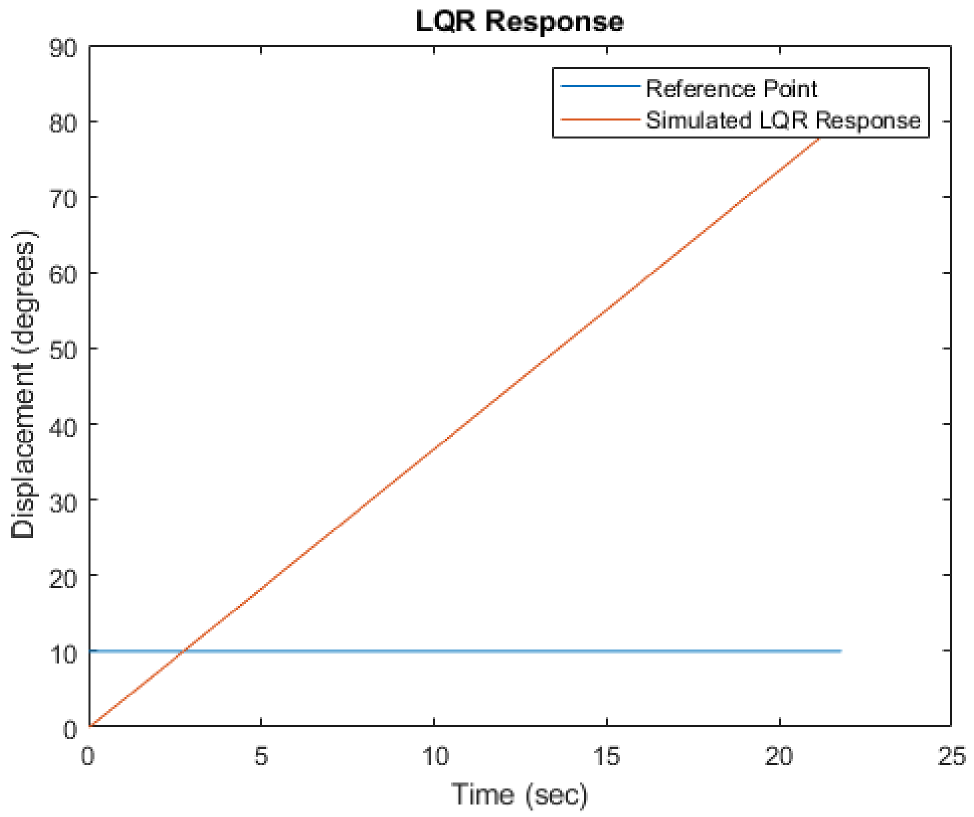


Figure 3.10: LQR response Q = identity matrix and R = 1

The Q and R matrices will be set to identity matrices as starting points and from initial testing, changing the values of R and the first and second eigenvalues of Q have little to no effect on the system. Changing the third eigenvalue of Q to 10^{11} decreases the time it takes for the system to stabilize as shown in Figure 3.11 below.

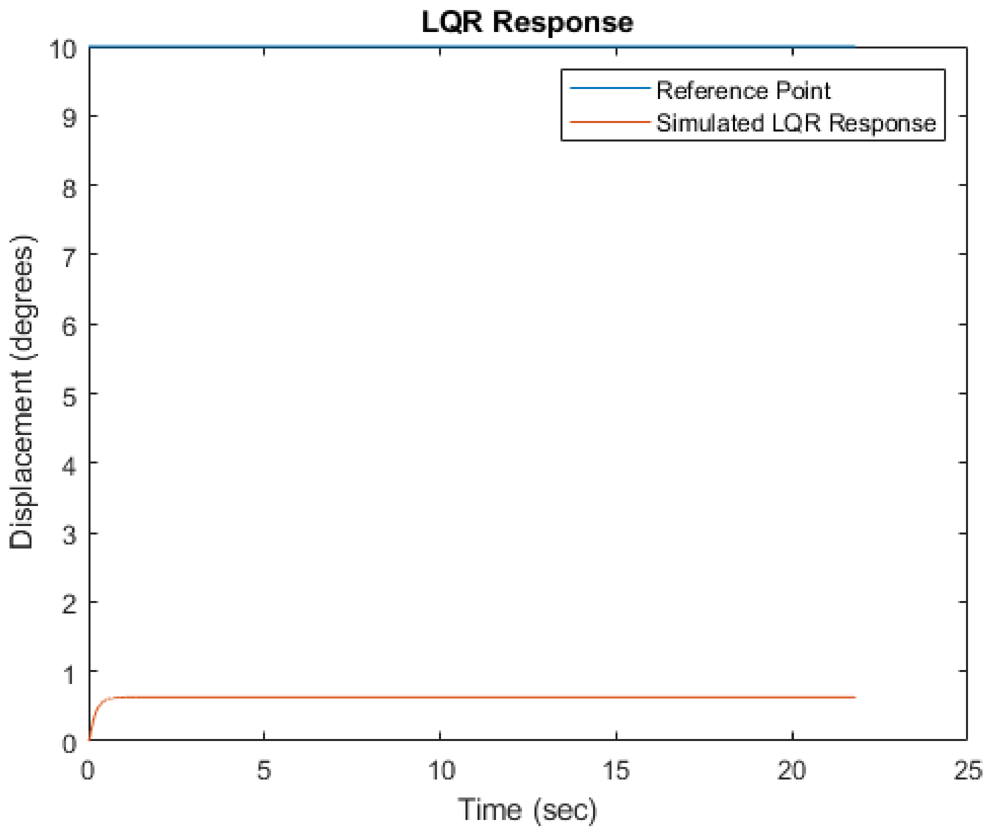


Figure 3.11: LQR response Q = modified matrix and R = 1

While the controller was able to stabilize relatively quickly, it could not achieve the desired reference value. This situation is called the steady-state error and may be caused by the reference signal being appropriate scaled compared to the state values. This can be remedied by implementing a pre-compensator after the reference signal. After implementing the pre-compensator appropriate gain value N of 15.867 to the reference signal, the LQR can successfully track the reference signal at various values.

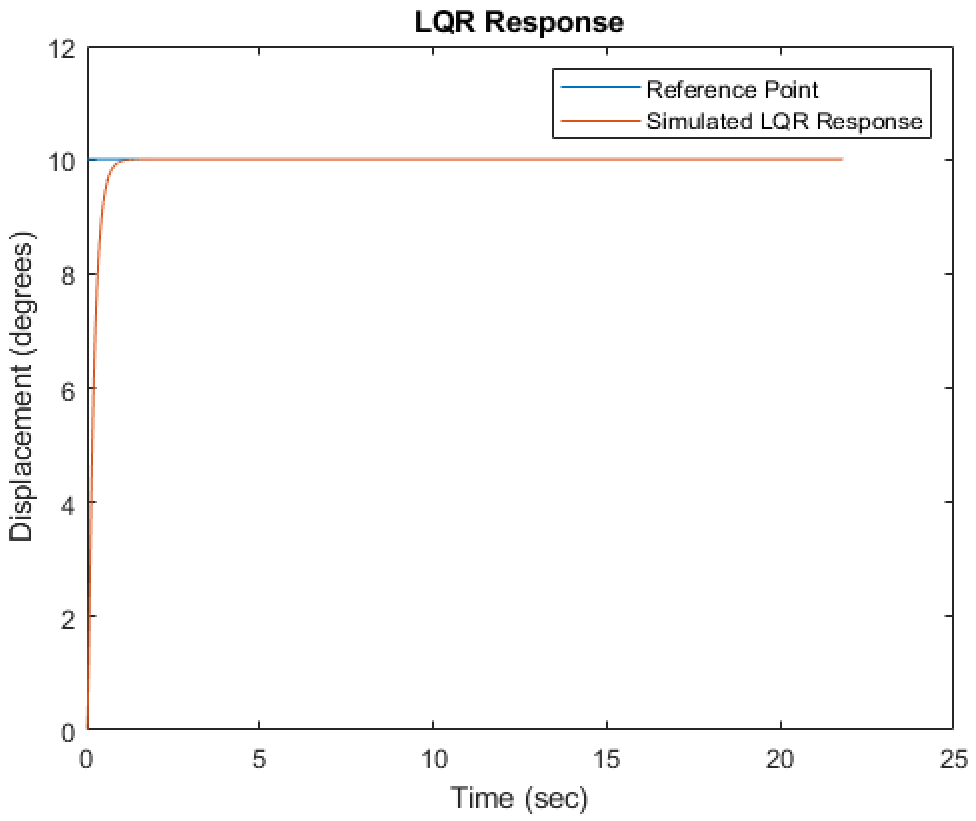


Figure 3.12: LQR response Q = modified matrix and R = 1 and N = 15.867

Comparing the two controllers show that the LQR controller does not overshoot compared to the PID controller; however, the LQR will require more accurate analysis of the model to provide the correct gain values.

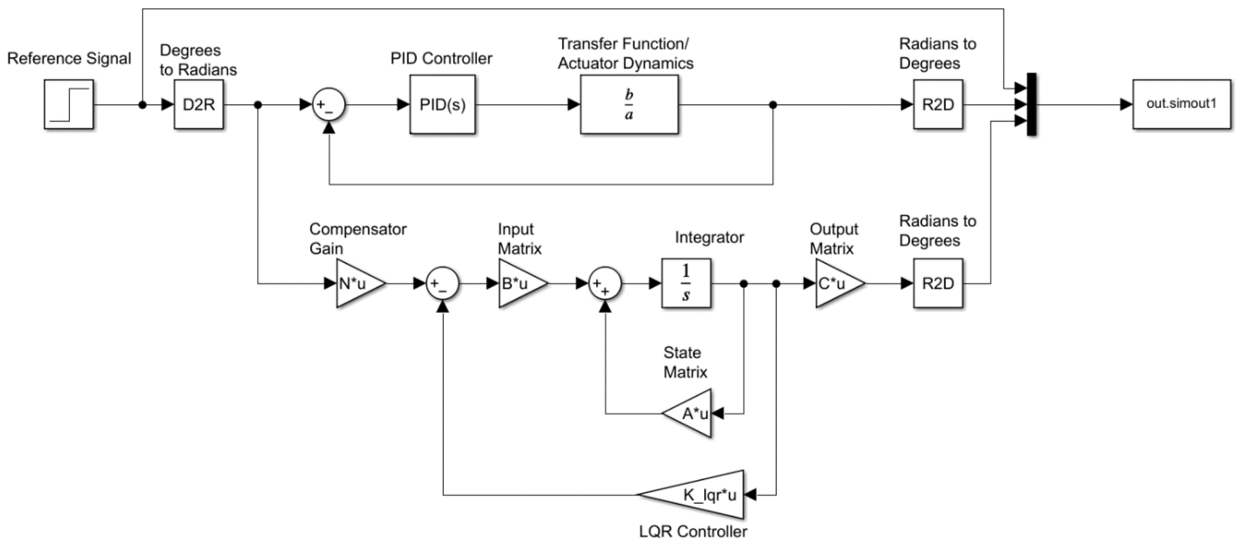


Figure 3.13: LQR and PID Simulink Diagram

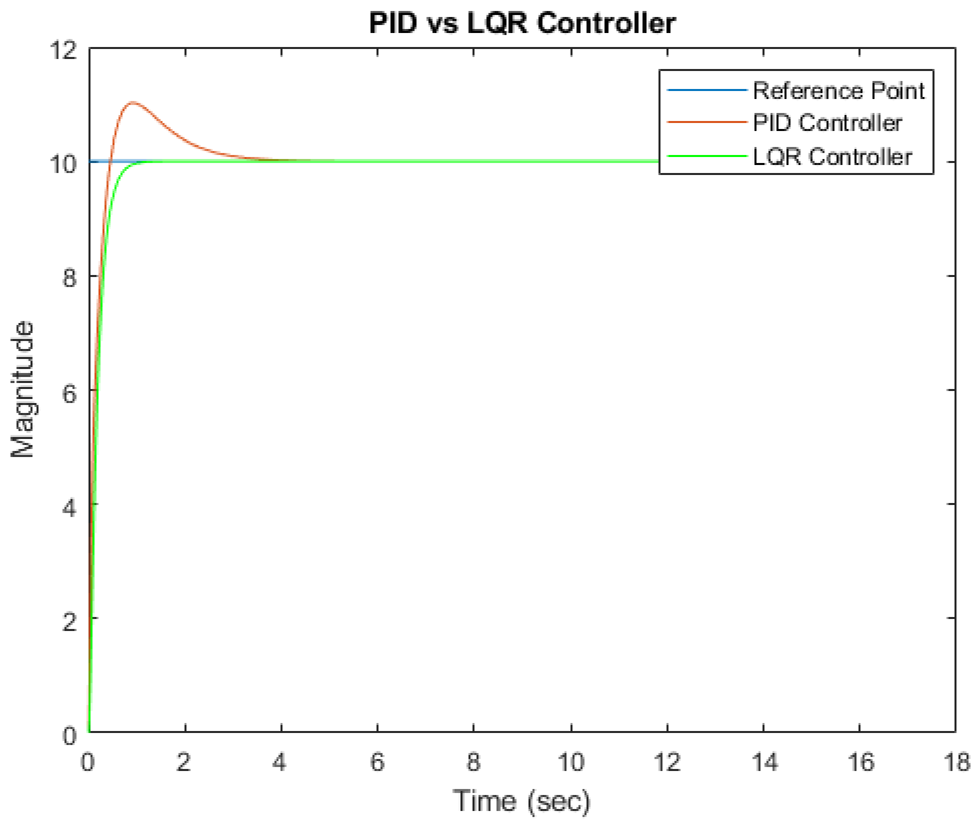


Figure 3.14: PID vs LQR controller response

Chapter 4: Development

While the core components of the satellite have been established, the satellite will require miscellaneous hardware to obtain and process the data. These pieces of hardware will only include the electronic components, which will all be powered by a 11.1V lithium-polymer battery. The L298N motor driver used for the motor has a supply voltage that allows current from to battery to power the support hardware.

4.1 Microcontroller

The voltage in the motor transfer function denotes the input signal sent from the main computer of this system, which is an Arduino microcontroller. A microcontroller is used for directly interfacing hardware and executing simple code interpreted in its firmware. The specific Arduino model chosen for this project is the KEYESTUDIO Mega Plus 2560 R3 Board, which is a clone of the Arduino Mega 2560 R3. This clone has 2 ampere output, allowing it to operate more sensors than the original. As satellites requires data compression and communication with ground base, an alternative computational framework may be required as the Arduino Integrated Development Environment platform is not equipped to handle multiple programs. The Raspberry Pi is a microcomputer with its own processor that can run multiple complex programs. While the Raspberry Pi has more processing power than the Arduino, its lacks the analog input/output capabilities needed to control the motor speed. A combination of both may be required for hardware management and software control. This would require additional programming and system integration that is beyond the given timeframe of this project, which requires only accurate modeling of the tracking performance of the system. However, commercially available microcontrollers and microcomputers may not be functional in space due to the high amounts of ionizing radiation, which imparts high amounts of energy into the electronics that could affect the signals or even permanently damage the transistors [21]. This can be mitigated by effective shielding using a radiation-resistance casing or a process called radiation hardening. This is achieved by changing the material of the electronic components to something that can resist radiation, as well as changing the transistor design to be less sensitive to noise. For Technology Readiness Level 6, which is a system prototype demonstration in a ground environment, the various subsystems will not take into consideration radiation resistance. When the system is retrofitted for spaceflight, the hardware will be re-evaluated for radiation resistance and the dynamics of the system will be adjusted.

4.2 Object Tracking Camera System

The tracking of this system operates by taking input from the camera system and feeding it through a closed-loop system to minimize the error between the center of view of the camera and the location of the tracked object. This camera system consists of a camera connected to a computer running a specialized algorithm that recognizes image patterns programmed by the user. In the field of computer vision, there are currently three design options available for this project:

- Commercially available computer vision devices
- Open-source computer vision programs
- Developing computer vision software

Of the three options, developing computer vision software from scratch would be the most unrealistic. While this method would theoretically allow for most customizability and there are guides on achieving accurate tracking, this is well beyond the scope of the field of aerospace engineering. The purpose of this project is to just identify and track debris by using the dynamics of the reaction wheel to keep the object at the camera's center of focus. As such, developing a computer vision code from the ground up would be extremely time-inefficient for this project. While open-source computer vision programs and supplementary tutorials are readily available that would provide a relatively high degree of customizability regarding object segmentation programming and camera specifications, they also fall outside the realm of aerospace engineering. However, this option will be reevaluated for integration with space-grade cameras. Currently, this project will be utilizing a commercially available computer vision device known as the Pixy2, primarily for its ease of setup and efficient tracking. This unit can be connected to a Raspberry Pi microcomputer or an Arduino microcontroller. Currently, the effective range of the object detection is 4 feet which can be extended with fine-tuning of the object identification. The Pixy2 Camera unit would be fixed to one end of the main body, which will be considered the X-axis of the body frame and will be aligned with the beam of the laser system. This would allow the beam from the laser and the center of view of the camera to remain relatively close to each other and minimize the hardware adjustments needed for the beam and the center of view to converge. The camera unit can be preprogrammed to detect certain objects before operation, which would require the camera to be connected to a computer as demonstrated below.

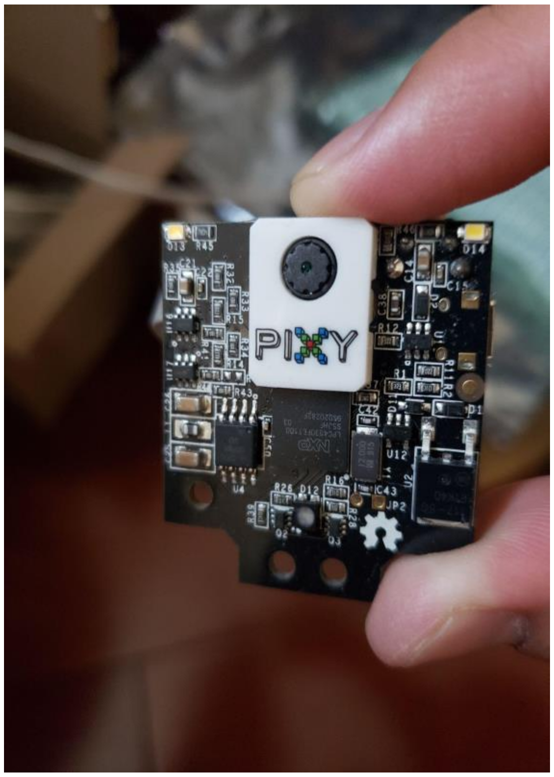
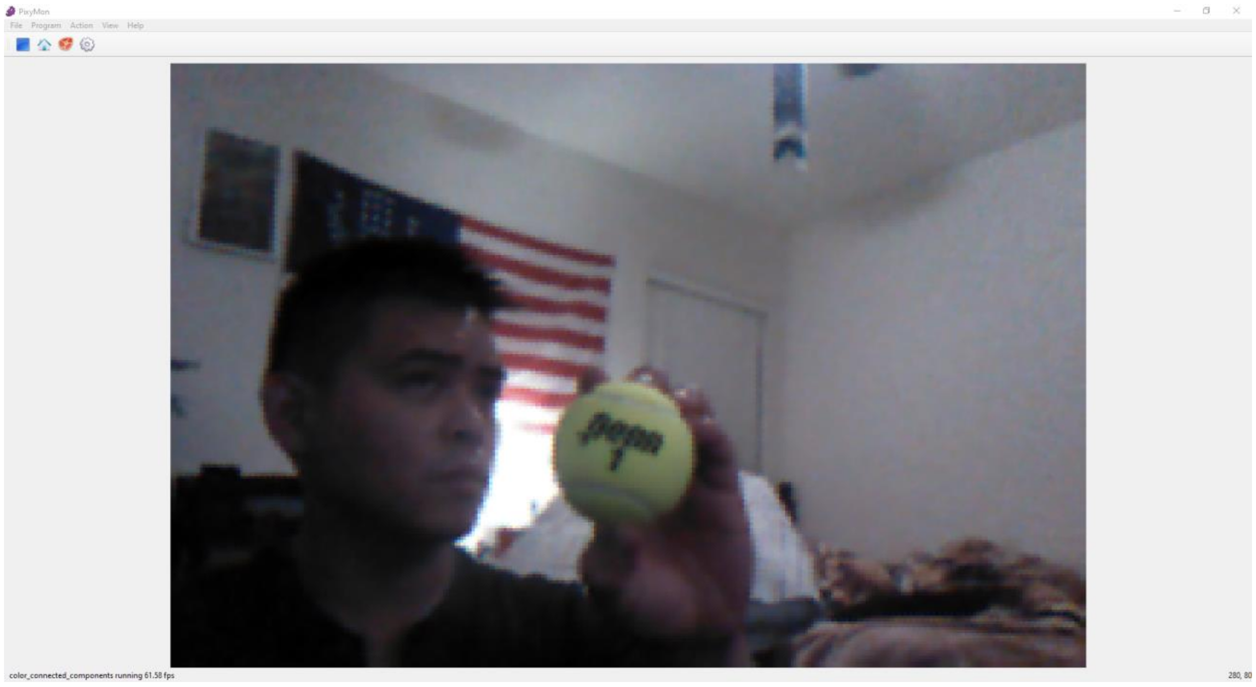


Figure 4.1: Pixy2 camera unit



```
color_connected_components running 61.58 fps
```

Figure 4.2: Live video feed

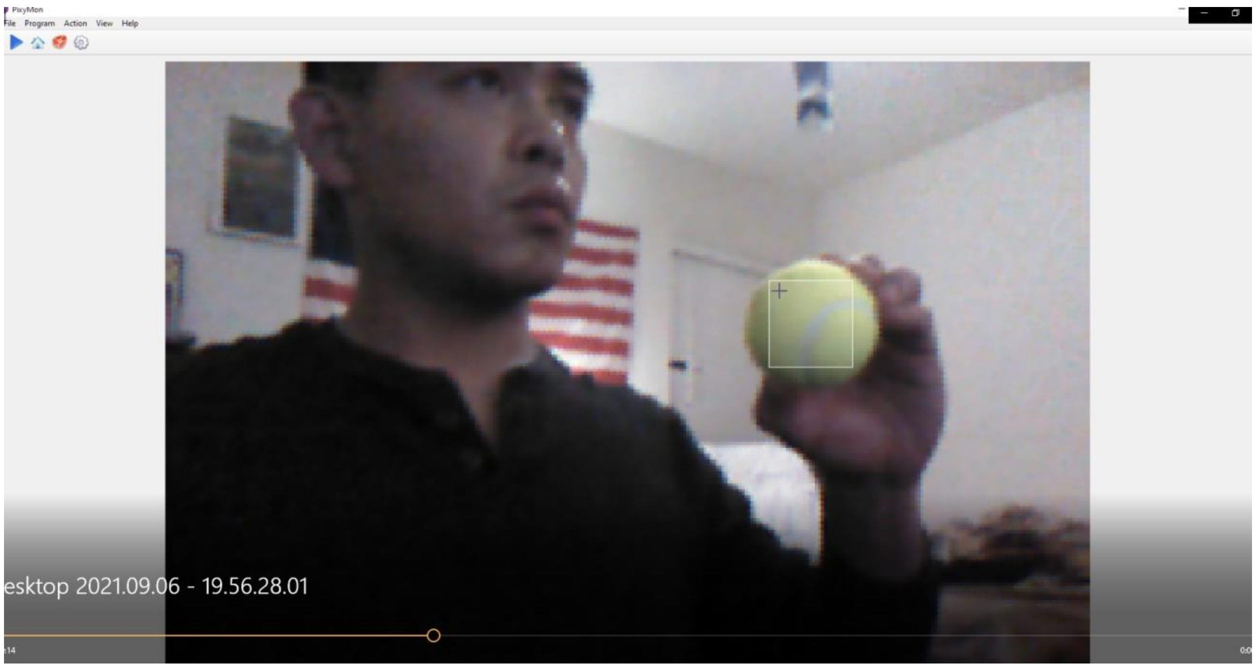

PixyMon
File Program Action View Help
esktop 2021.09.06 - 19.56.28.01

Figure 4.3: Selecting object to be detected

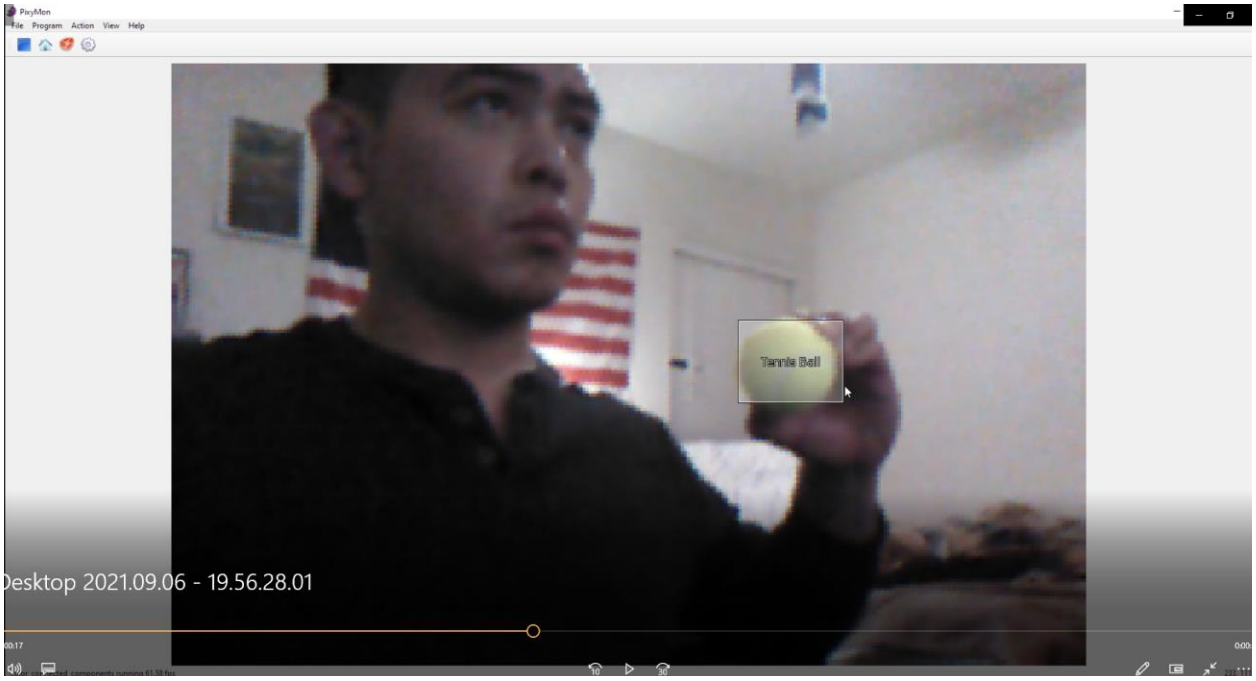

PixyMon
File Program Action View Help
Tennis Ball
Desktop 2021.09.06 - 19.56.28.01
00:17

Figure 4.4: Segmentation of object

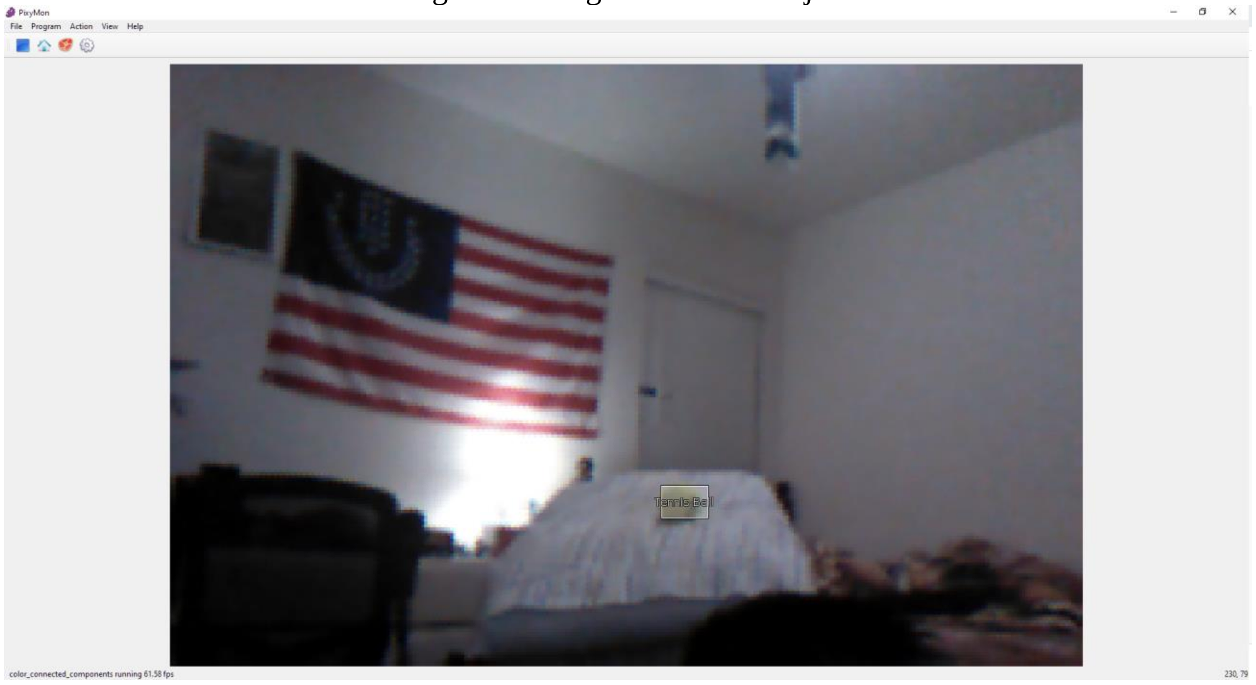

PixyMon
File Program Action View Help
Tennis Ball
color_connected_components running 61.58 fps
230, 79

Figure 4.5: Effective max range of object tracking without tuning

In its default setting after initial setup of cataloging the tracked object, the Pixy2 camera may register numerous false positives. This can be remedied by configuring the camera settings and maximizing the block filtering, which allows for accurate readings of the tracked objects at the cost of increasing the response time of the camera. However, from testing, the increased response time is negligible.

4.3 Inertial Measurement Unit

To test the validity of the simulations, comparisons will be made between the simulations and the tests performed on the physical system. This will require a device that can measure the system’s change in the yaw direction relative to its initial position. The GY-521 Inertial Measurement Unit will be used to measure the system’s angular velocity and relative angular displacement but directly connecting the GY-521 to the Arduino will only yield raw unreadable data. Specialized libraries called the “I2Cdev” and the “MPU6050” by Jeff Rowberg would need to be installed in the user’s Arduino IDE program to allow the GY-521 to access the Arduino’s serial clock and data lines [22]. The MPU6050_DMP6 example would provide the function necessary to convert the raw data into roll, pitch, and yaw in degrees. One issue that arose and was acknowledged by the developer was the lack of validation of the yaw angle as the yaw provided from the code will drift over time with no means to correct with. The software will set the location of its initial activation as reference point but will not have means to validate the angular displacement. This can be corrected by using a 9-degree of freedom IMU as this IMU will have access to a magnetometer. This device will use earth’s magnetic field as a reference frame to validate its yaw position; however, the code would need to be modified to incorporate this device. For short-term testing, the GY-521 IMU will provide sufficient feedback of the angular displacement of the system. As the system activates, the IMU sensor will take its initial orientation as its zero point and further movement of the system will be indicated in the code. To validate the tracking performance, the IMU sensor will be aligned directly behind the camera to provide feedback of the system’s yaw movement.

Chapter 5: Hardware Test

5.1 Arduino Code Process

As the system utilizes an Arduino microcontroller, the code for the system will be based on Arduino IDE. This section will explain the functionality and logic behind the code. Parts of the code have been based on guides made publicly available by users on GitHub. This project is done for educational purposes and no profits have been made from this endeavor. The process behind this code is based on “if/else” logic statements regarding the position of the ball relative to the camera’s center of view. The relative ball location will be fed to the microcontroller where if it exceeds a buffer area called the deadzone, the motors will activate to decrease the distance between the center of the ball and the center of view of the camera until the distance is below that of the specified deadzone. Below is a flowchart diagram of the process behind the code.

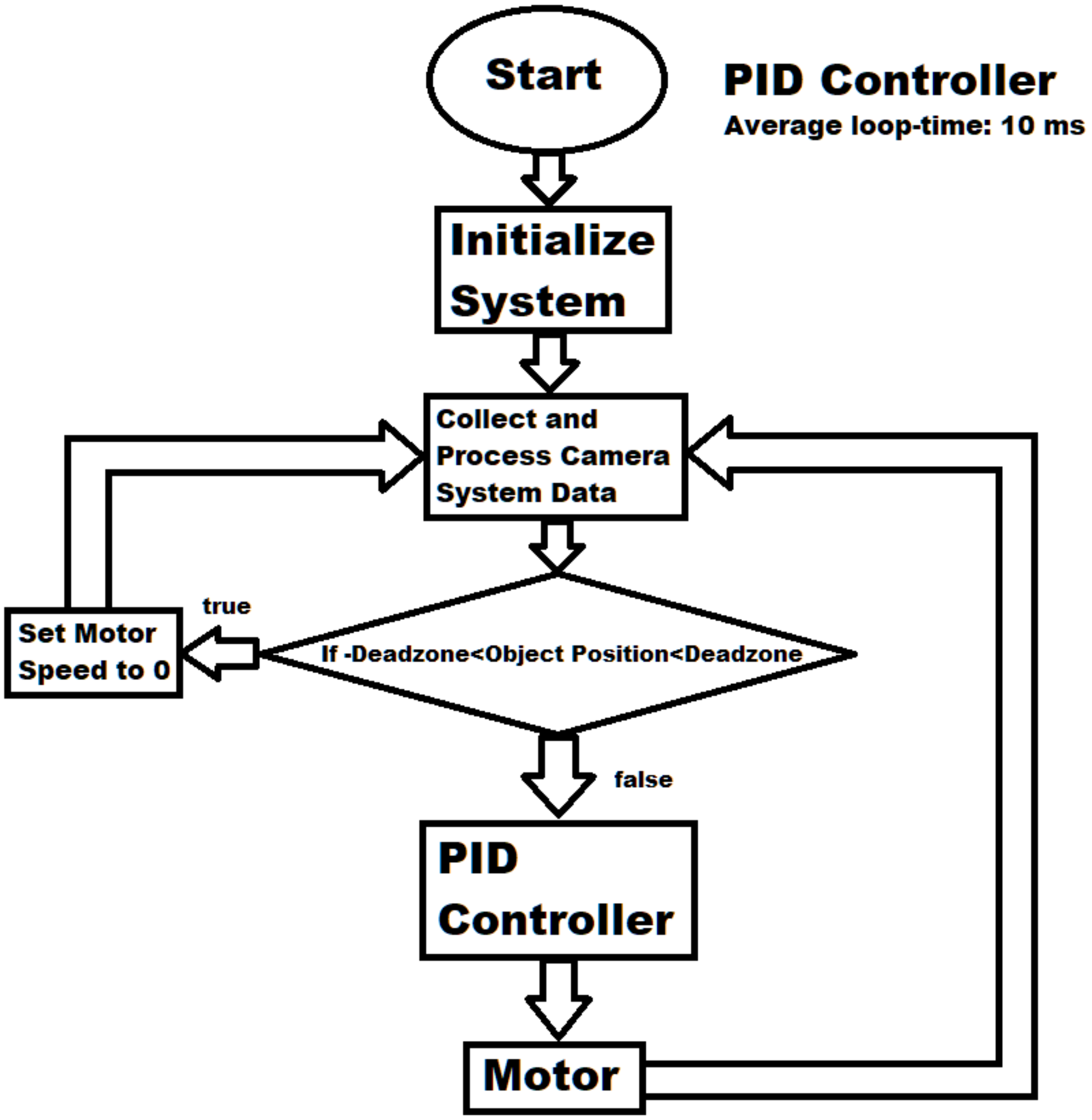


Figure 5.1: PID controller flowchart

The camera system used for this satellite is the Pixy2 Camera, in which the developers have posted documentation and code for on their website [23]. For organizational purposes, the section of the code that operates the logic of the camera system will be placed in a function at the end of the code. After installing the necessary libraries into Arduino IDE and configuring the camera to track a specific object, the code can activate the Pixy2 by calling pixy.ccc.getBlocks(), which grabs any objects listed in the camera's database and places a box around them. Due to the camera ratio, this will return a value from 0 to 320 along the horizontal axis, which accounts for a field of view of approximately 60 degrees. This can be normalized from -160 to 160 by using the map function, which allow the center of view of the camera to be considered the 0 point to be utilized later by the deadzone feature. Once mapped, anytime the camera detects the desired object, it will output a value from -160 to 160 from this function to the main block of code.

The value obtained from the camera system will be placed in the void loop block, which will perform its assigned function continuously. To avoid unnecessary spinning of the satellite, the command pixy.ccc.getBlocks() will be used to activate the camera to detect any objects that were listed in its database. If there are no objects detected, no signal should be sent to the motor. This can be achieved by using the conditional requirement command "if (!pixy.ccc.numBlocks)", followed by digitalWrite(out_B_IN4,LOW), digitalWrite(out_B_IN3,LOW), analogWrite(out_B_PWM,0). The digitalWrite commands sets both motor polarity to LOW meaning that the corresponding voltage is set to 0. The analogWrite command is used to write the signal to the motor driver that ranges from 0 to 255. If the object is detected but in in the center of view, the system will change one of the motor polarities to HIGH, so that current is able to flow through the voltage differential. The value of the analogWrite command will be determined by the PID function. While there are many PID libraries on Arduino, the PID library used in this system is by Brett Beauregard who based the PID on this equation established in the Closed-Loop Dynamic Model Stability Analysis [24].

$$Output = K_p \cdot error(t) + K_i \cdot \int error(t) \cdot dt + K_d \cdot \frac{d}{dt} error(t) \tag{5.1}$$

This function requires the users to set values for the Input, Output, and Setpoint as well as input the K_p, K_i, and K_d values, in which the values obtained from the MATLAB Simulink autotune can be used. The Setpoint is the value that the PID is attempting to achieve, which will be 0 or the center of the camera. Using the value of the distance between the centerline and the object as the Input, the difference between them will be the error that the PID will solve. The error is calculated as the Setpoint minus the Input. Due to the 8-bit binary value of the Arduino, the values of Output only range from 0 to 255. If the Input is a positive value, the error would be calculated as a negative value, which the Output would consider as 0. This results in the motor not spinning a certain direction. This can be remedied by placing setting Input = -Input, which turns the error into the positive value, allowing the PID's Output variable to be a value from 0 to 255 depending on the object's distance from the center of view of the camera.

As the center of view of the camera approaches the object, the signal received from the PID will continue to decrease. However, due to the fluctuations in the object detection as well as the movement of the satellite, the PID might send signal to the system that would cause the satellite to overshoot for the object and become unstable. To prevent this, offset boundaries, called deadzones, will be set to the sides of the center of view of the camera system. These deadzones serve as a switch for the PID, by which, should the view of the camera be focused on an object at an acceptable point that the user considers, the switch will cease all motor function until the tracked object moves again. From the Pixy2 camera's documentation, its horizontal view range is 320 units in the system, which is approximately 60 degrees. The deadzone will account for 10 percent of the total field of view, which is roughly 6 degrees or 32 units. The loop logic of the code operates under if else statements. If the camera detects an object, it will send its position relative to the camera's center of view as the Input to the microcontroller. If the Input is greater than the deadzone, that means that it is too far to the right and the system must spin to the right. If the Input is less than the negative deadzone, that means that it is too far to the left and the system must spin to the left. If the Input is directly in front of the camera, then the motor ceases operation until the Input shifts out of the deadzone range.

In the previously mentioned GY-521 library by Jeff Rowberg in the Inertial Measurement Unit, an example code is included that allows the user to customize the packets to output certain variables [23]. Of those variables are the yaw, pitch and roll which are included in the packet "OUTPUT_READABLE_YAWPITCHROLL". Activating the example code in Arduino's serial monitor prompts the user to type any character in the command window to begin the reading of the IMU sensor. The software that will be used to record the IMU sensor is not capable of sending an input to the Arduino, thus the section of code requiring user input can be deleted. In the void loop() section, the commands to obtain the roll pitch and yaw can be copied and pasted into the void setup() section to establish an initial point of reference for the system. A ten second delay will be placed after this to allow the user to correct the orientation of the system and ready the testing site.

Data is normally outputted from the system through Arduino's serial monitor; however, the serial monitor does not allow for saving the data in a readable file format. Testing the system will requires a 3rd party addon named ArduSpreadsheet [25]. This addon acts as a second serial monitor that allows the user to save data as a CSV file which can also be read as an XLSX file which is readable on Microsoft Excel.

5.2 Testing and Analysis

The tracking performance of the physical system will be tested against those of the MATLAB Simulink model to validate the tracking performance of the controllers. Of the two controllers, PID controller libraries are more readily available and simpler to implement compared to the LQR controller. Multiple tests will be conducted in a manner such that human error is kept to a minimum and the average of these trials will be compared to that of the controllers designed in the simulations. For the PID controller implemented in Arduino, the values used for the K_p, K_i, and K_d values are 15.142, 14.922, and 0.954 respectively, which are obtained from the using the autotune feature of the PID block in MATLAB Simulink. The test will involve the system being placed on top of a large, marked sheet of paper. This sheet will act as a placemat for the system and will have a centerline through the length of the sheet as well as another line on one end of the sheet that is tangent to the centerline. The intersection between these two lines is where the center of the center will be placed. As with the case in the simulations, the tracked object will be placed 10 degrees from the camera's center of view, which is aligned with the placemat's centerline. Once the system is powered, set on the placemat, and connected to a computer, The system will wait until a command from the user is sent from ArduSpreadsheet. Once the user initializes the system from ArduSpreadsheet, the user will have 10 seconds to ensure that the system is aligned with the placemat's centerline. After the delay, the motors will activate to align the camera with the tracked object. 11 test runs were completed and compiled where the results were compared to the PID of the simulations.

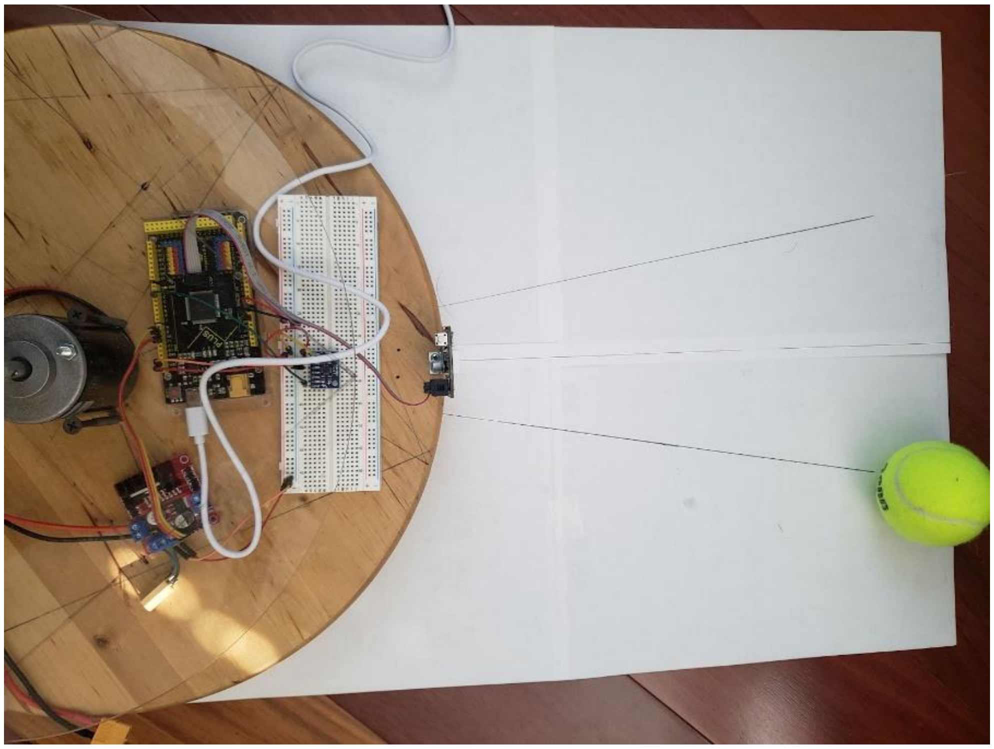


Figure 5.2: Testing pad

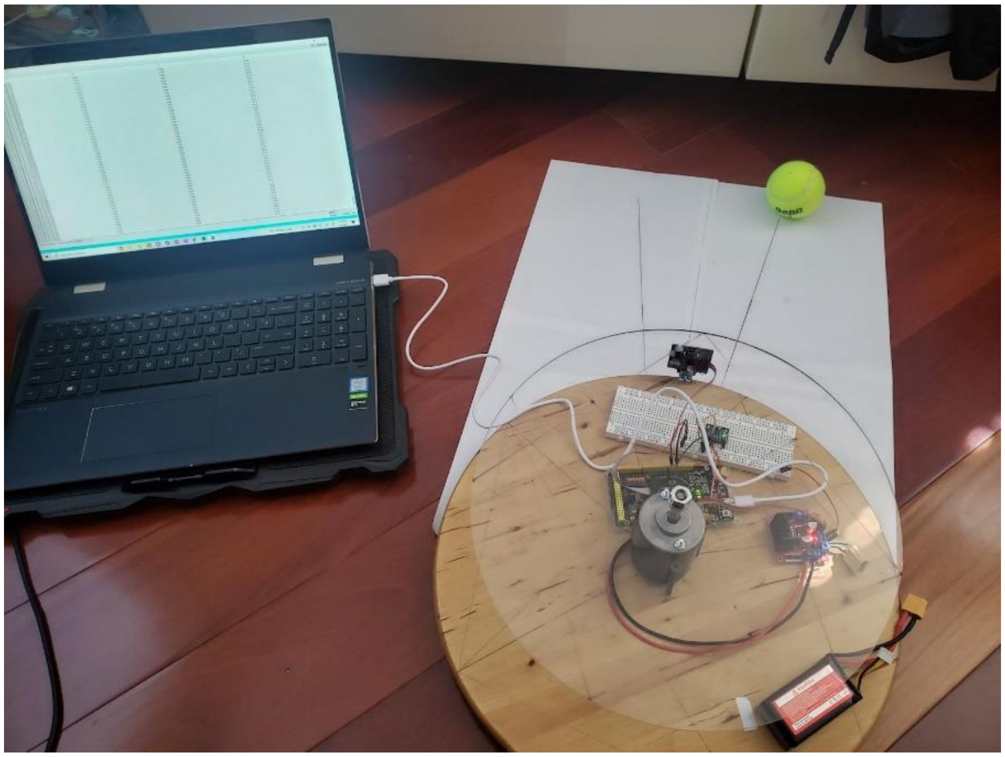


Figure 5.3: ArduSpreadsheet recording

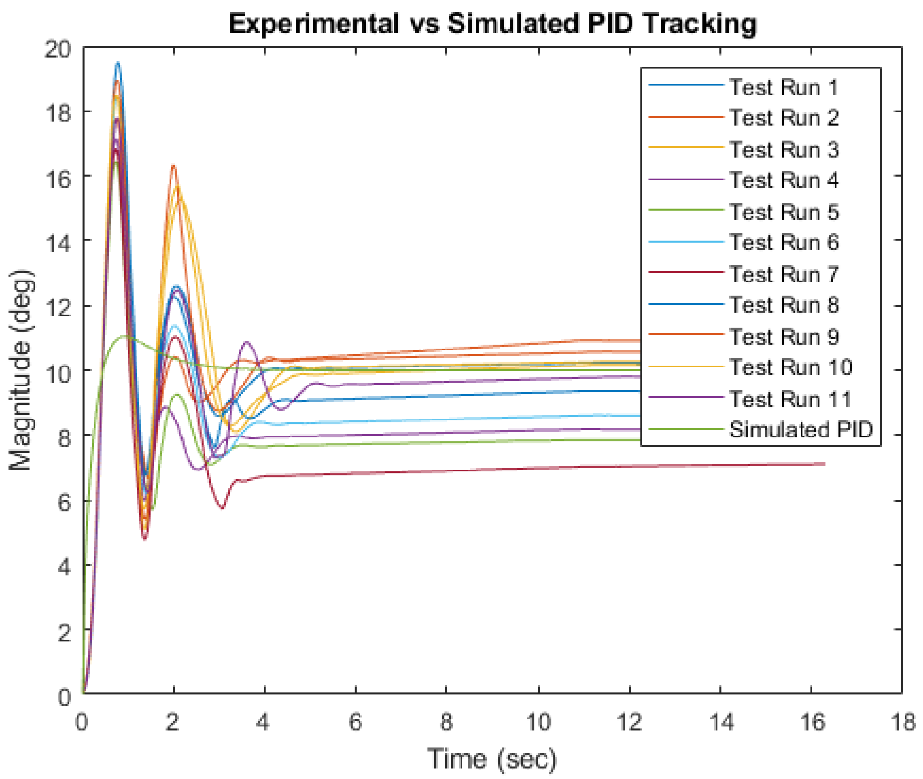


Figure 5.4: Experimental vs simulated PID tracking comparison

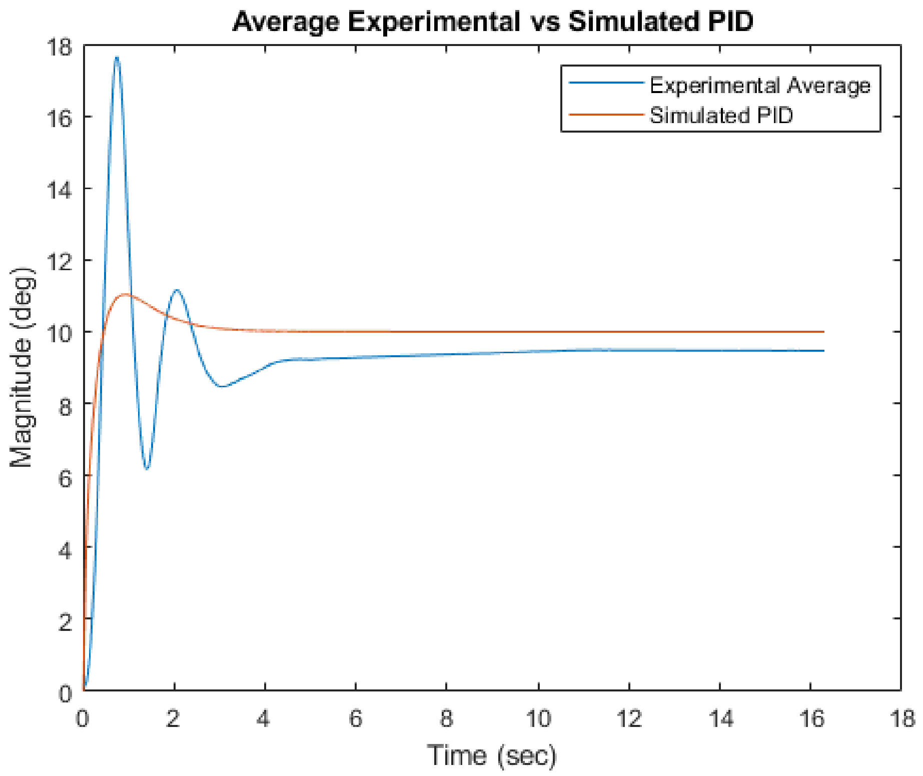


Figure 5.5: Average experimental vs simulated PID tracking comparison

From the tests, the average of the tests will plateau at roughly 9.48 degrees; the largest deviation of which comes from test run #7, which stops oscillating at 6.7 degrees but steadily rises to 7.11 degrees. This steady rise is noticeable in all the tests performed and may be attributed to the hardware limitations of the GY-521 IMU, which cannot validate the yaw axis without the use of a magnetometer. The initial oscillations can be attributed to the deadzones established in the code where upon activation, the system will spin the reaction wheel to face the tracked object. Once the system's center of view reaches the deadzone, the Arduino will cease sending signal to the motor; however, the reaction wheel will continue to spin due to the leftover angular momentum, causing the system to continue turning past the deadzone. This deadzone measures 10 percent of the camera's field of view away from the center of the tracked object, which is a margin of error of 6 degrees for the deadzone. Video recordings were taken of the system during the tracking process. Below are still images of the system during the peak of the oscillation.

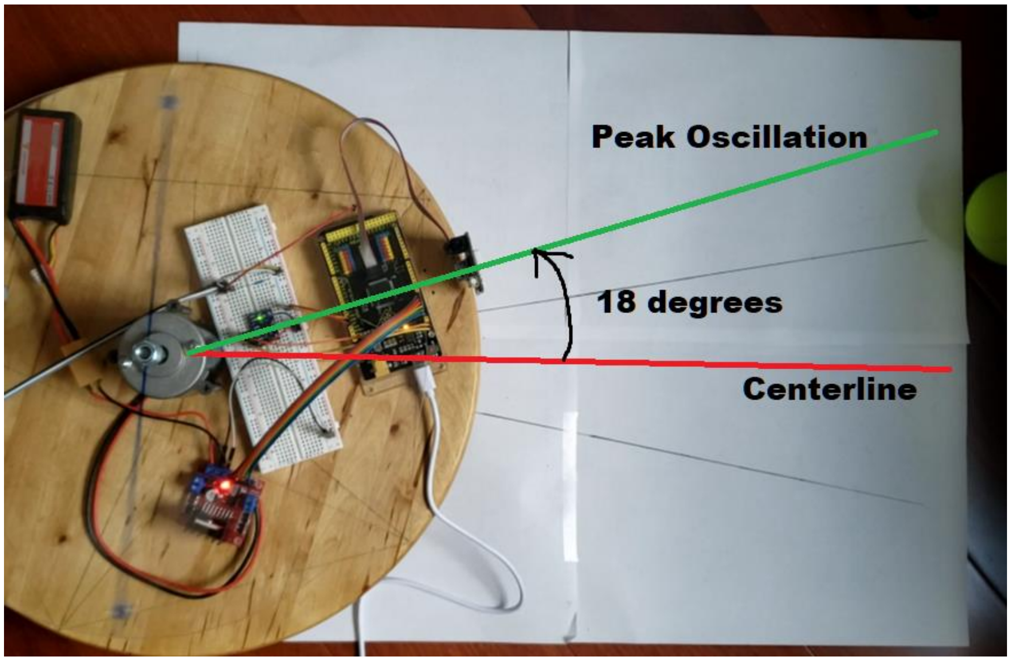


Figure 5.6: Experimental oscillation peak 1

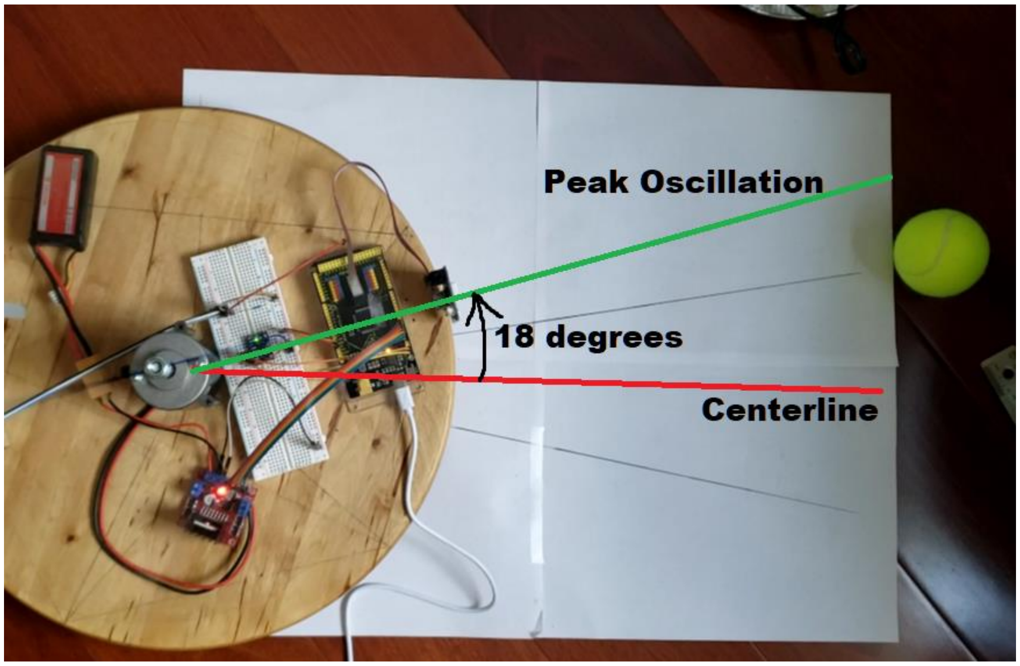


Figure 5.7: Experimental oscillation peak 2

Once the camera has detected that the system has rotated 6 degrees past the tracked object, the motor would be activated again to maintain contact with the tracked object. While this may return the system's center of view to the object, due to the leftover angular momentum, the system may continue to spin outside of the deadzone, causing frequent oscillations. Below is plot of the system response of one of the tests that was nearly able to track the center of the object.

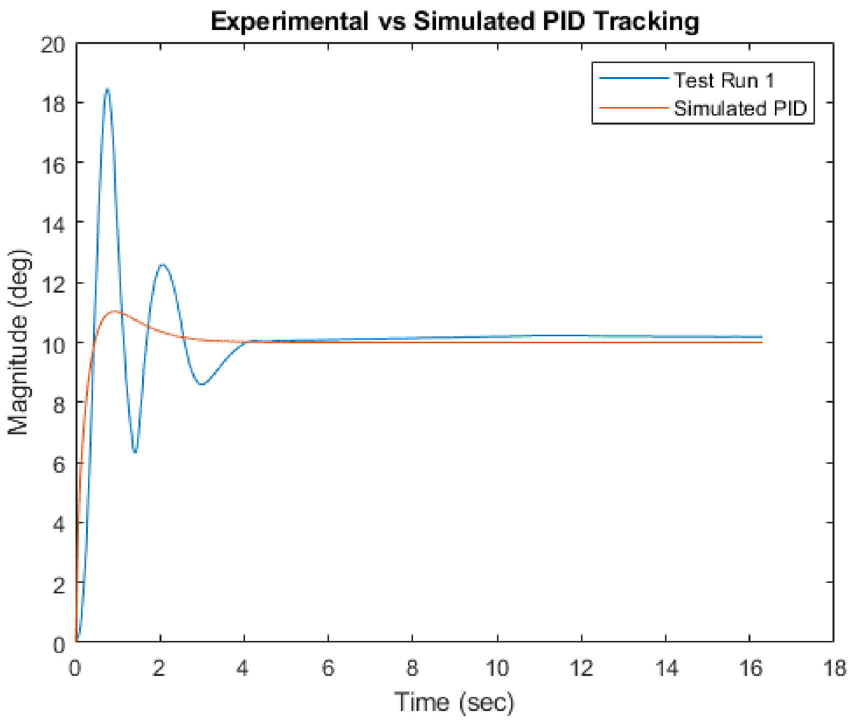


Figure 5.8: Test run 1 vs simulated PID tracking

In the graduate project report, “Design and Testing of a Nanosatellite Simulator Reaction Wheel Attitude Control System” by Frederic William Long at Utah State University, Long designed a satellite that has four reaction wheels arranged in a pyramidal configuration [11]. The purpose of this project is to design an attitude controller for pointing communication antennas and other instruments using reaction wheels. This system is further developed than that of the space debris laser sweeper in that it can be controlled along the roll, pitch, and yaw axis using a PID controller. It differs from the space debris laser sweeper in that it uses Euler angles as outputs for the roll, pitch, and yaw. The values chosen for the PID are also dependent on the scalar natural frequency instead of fixed values determined from the MATLAB Simulink autotune block. In one of the tests performed in the nanosatellite simulator, the yaw axis was tested by itself to reach a reference point of 90 degrees every 10 seconds. This exact test cannot be performed by the space debris laser sweeper as it relies on the camera system which as a max field of view of 60 degrees; however, the tracking response of the reference value can be compared. In the test plot show below, the measured tracking response does initially oscillate much like the space debris laser sweeper; however, the overshoot is far smaller at roughly 42%. The initial overshoot at the roughly 1 second mark to get to 0 is much closer in shape to the space debris laser sweeper, which the authors measure at 48%. In the benchmarked report, the methodology for calculating the overshoot was not specified. To compare the results between the experiment and the benchmark, the overshoot percentage will be based on the percent difference between the actual displacement to the peak of the oscillation and the intended displacement. Starting at 120 and reaching the peak of the oscillation at -80, the overshoot percentage is 67%. Using the same methodology for the experimental data, the overshoot is 80%, meaning that the initial overshoot of the space debris laser sweeper is comparable that the initial overshoot of the benchmark as show below.

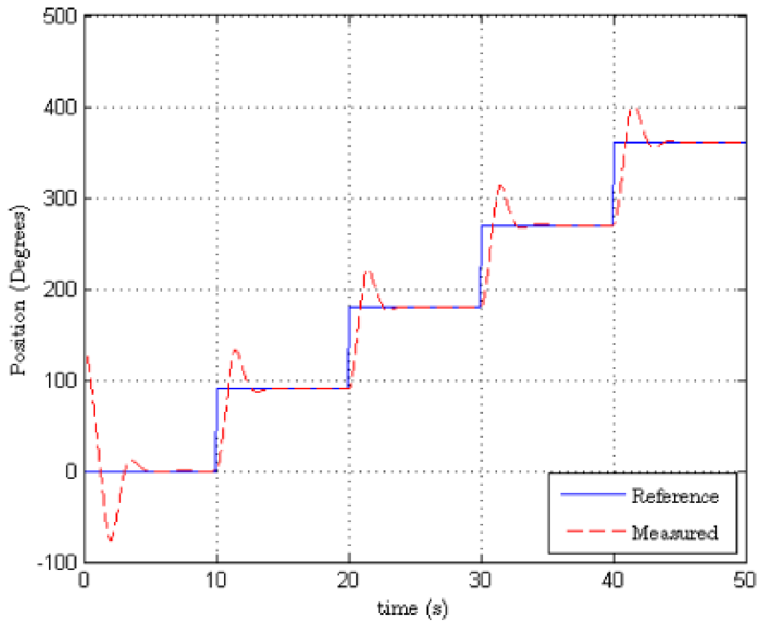


Fig. 6.11: References and measured position in yaw with $\omega_n = 2rad/s$ for the no trajectory test

Figure 5.9: Benchmark tracking response

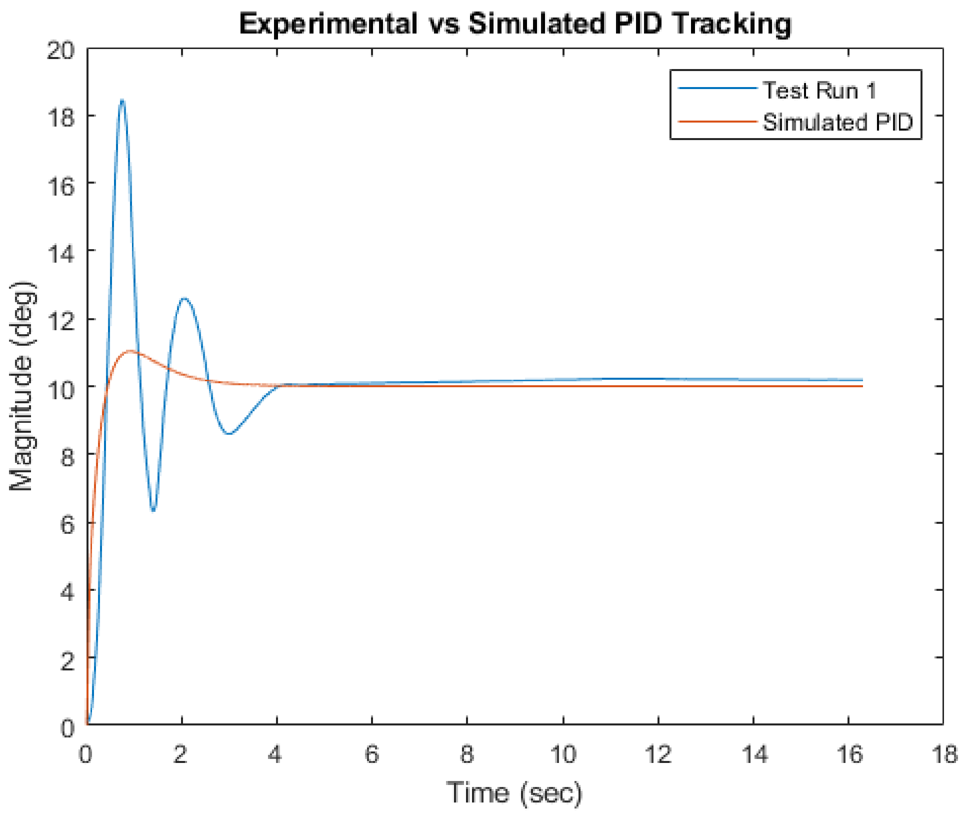


Figure 5.10: Test run 1 vs simulated PID tracking

Even the test that was able to track the object the closest during steady state included oscillations during the initial tracking process. This is due to the leftover angular momentum and the fact that there is no way to directly control the angular displacement of the motor shaft. Alternate motor choices have been considered but would requires drastically different hardware and programming. Stepper motors allows for direct control of the motor shaft at the cost of speed and efficiency, while DC motors with encoders may require additional hardware and software to regulate the angular velocity of the motor shaft. Alternatively, the PID values can be adjusted to minimize overshoot, but may come at the cost of response time. A method for modeling the angular momentum of the reaction wheel may be required to replicate and optimize the tracking performance of the system.

Chapter 6: Conclusions and Future Work

6.1 Completed Work

The purpose of this project is to design and build a working proof of concept for a space debris laser sweeper, of which the most important aspect is the ability to detect and track objects in a low gravity environment to simulate orbit. While modifications will be required for the hardware to achieve spaceflight, the software is able to perform object tracking in the yaw direction at speeds comparable to that of the optimized simulations. Assuming the system was redesigned for gravity gradient stabilization, a second motor controlling the roll or pitch axis can be added using the same code, albeit with difference values to account for gravity. The camera system, which was initially thought to be the most difficult aspect during the conception of this project, was simple to implement due to the developers behind the camera system providing detailed guides on how to program and interpret data from the camera. Controlling this system with reaction wheels would require motors more powerful than anything the base Arduino Mega 2560 board can output. The method of mounting the reaction wheel to the motor shaft was also taken into consideration as most commercially available motors have a shaft that is a straight cylinder, which could not secure the reaction wheel to the shaft. As such, a 12V brushed DC motor with a threaded motor shaft was chosen for this system. With the frame, motor, and reaction wheel chosen, these parts formed the base for the system to be modeled.

Concurrently with the developing the code for this experiment, the system was also modeled in MATLAB Simulink to provide simulations of the optimal controller design within the given constraints. This required accurate modeling of the motor, reaction wheel, and platform until the final transfer function of the system was established in equation (2.15). Using this transfer function, the PID and LQR controllers were able to be designed. The values provided from the PID were implemented into the Arduino code. Using an IMU sensor to measure the angular displacement of the system, the tracking performance using the PID controller was recorded and compared to the simulations. While there were oscillations during the initial step of the tracking, this quickly dampened out and the system was about to track the object with a margin of error of 6 degrees. This margin of error will be an issue for tracking at longer ranges; however, the camera system may not be able to track objects at those ranges. During the transition to high technology readiness levels, the hardware will be reevaluated and may be swapped for parts rated for spaceflight.

6.2 Challenged Faced

There were several iterations of the transfer function of the system, the most rudimentary of which is only the conservation of angular momentum equation between the reaction wheel and the platform. This led to research on the process of converting electrical voltage to angular velocity. Of the equations that make up the transfer function of a motor, it was difficult to find information regarding the physical characteristics of the motor, particularly the viscous friction. Some research reports label the viscous friction as 0 while other reports provide a very small value for the viscous friction coefficient. This research of the characteristics of motors led to equation (2.13) being viewed as the transfer function of the system. Unfortunately, it was discovered that the output of that transfer function was labelled incorrectly as its output is the angular velocity instead of angular displacement. While this issue could be resolved in the

Simulink block diagram by adding an integrator at the end of the transfer function block, this solution would not work for the LQR block diagram as demonstrated in figure 3.25. To correct this transfer function, the angular velocity is converted to angular displacement via Laplace transform, which increases the rank of the system from two to three states as shown in equation (3.4).

As mentioned in the analysis of the test section, when the system tracks the object, once the object falls within range of the camera's center of view, the Arduino ceases sending signal to the motor; however, the reaction wheel is still spinning due to leftover angular momentum. This, in turn, affects the platform and will cause the system to oscillate. There have been measures to mitigate this by implementing a braking system in the code, which is achieved by briefly switching the polarities as the object nears the system's center of view. Unfortunately, this method proved extremely unreliable as the sudden turn in the opposite direction forces the platform to turn much farther than anticipated. Results were too inconsistent to make this method viable. The other optioned being explored now is the used of stepper motors or encoded motors as these motors have specialized hardware that allow for feedback of the correct angular velocity of the reaction wheel. In theory, this can be combined with a PID controller to output the necessary signal to minimize overshoot and prevent oscillations.

6.3 Future Work

Of the controllers designed for this system, only the PID controller was able to be implemented and tested due to the vast third-party support for the Arduino. The same cannot be said for the LQR controller support as LQR controllers are inherently more complex than PID controllers due to the matrix multiplication involved. As such, there are no LQR controller libraries available for Arduino, which means one would have to be developed from scratch. However, if the simulations comparing the two controllers are accurate to the physical tests conducted, the LQR would have greater tracking potential than that of the PID controller. The simulations show near instant tracking and no overshoot when it was used which are the biggest issues when choosing the PID values as demonstrated in the Simulink autotune feature. Implementing the LQR control system in the microcontroller would involve calculating each state through an iterative process, similar to how PID calculation are performed, albeit with multiple values for each iteration as opposed to one. This may require advance computing power beyond Arduino's ARM processor. Below is a recreation of what the LQR code operation could be like.

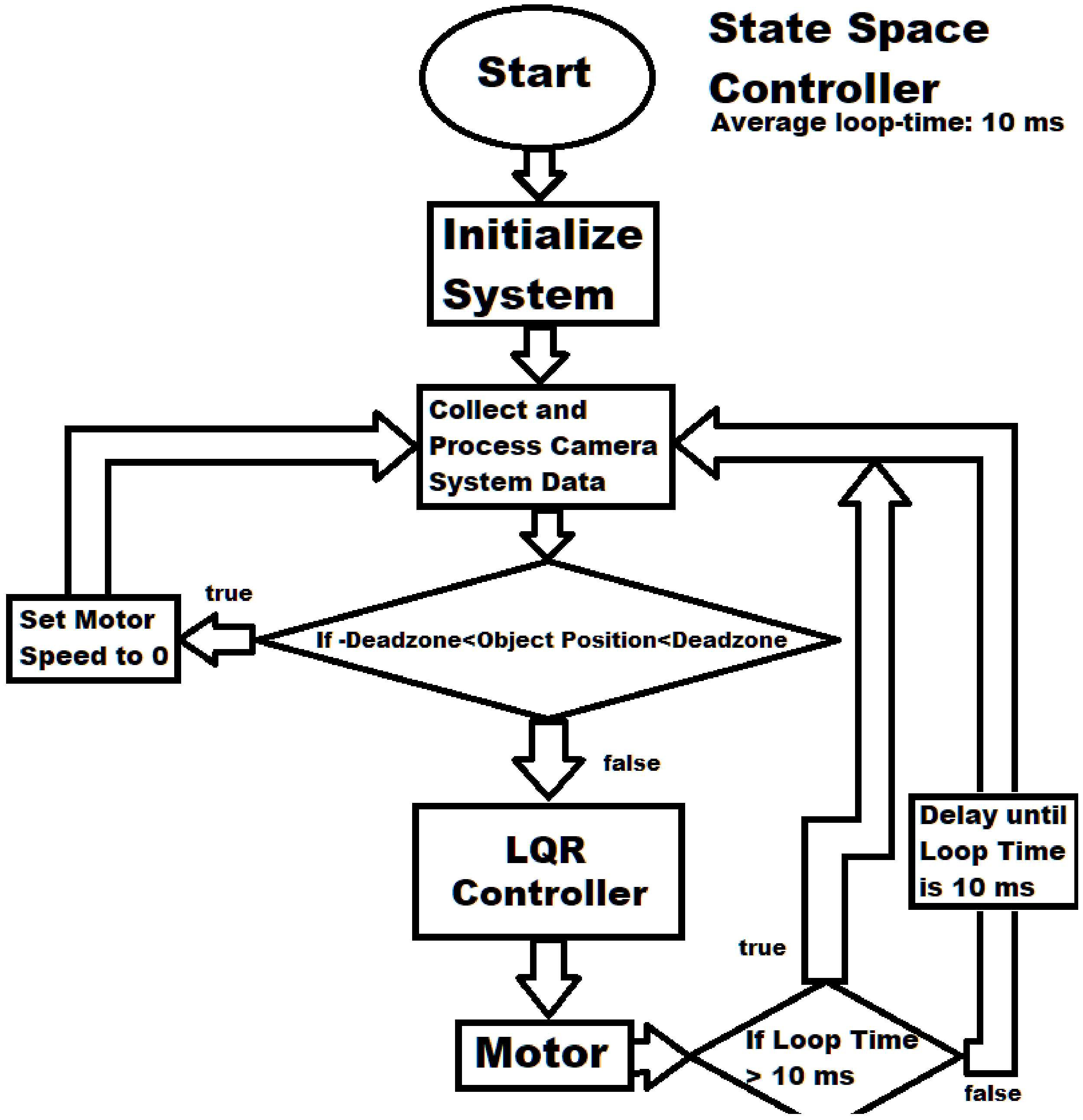


Figure 6.1: LQR controller flowchart

The equations to be used in the microcontroller would be the state equations which were derived in equation (3.9) and (3.10). Using a discrete time step of 10 milliseconds, the LQR controller could output the angular displacement, angular velocity, and the current draw from the state equations for each iteration. Further testing will need to be done to validate this process; however, if fully realized, this type of controller would be ideal for the space debris laser sweeper in that it would allow for multiple reaction wheels to be controlled with single controller, whereas multiple PID controllers would be required for each reactions wheel. This saves on energy usage which is the main cost function that would be divided amongst the guidance navigation and control, the laser system, and the computer. While the current PID system is more than capable of tracking objects, the overshoot of the tracking could potentially lead to the system losing track of objects moving at high speeds. The LQR controller would be able to take into consideration its current angular velocity and adjust its speed if necessary to maintain line of the sight of the tracked object.

Despite the purpose of the laser sweeper being to eliminate space debris using a high-powered laser, the actual laser will be the last subsystem to be developed in the overall system. The camera and control subsystems take greater precedence in the development of this satellite. While computer vision is a rapidly growing field, this analysis was mainly focused on the control/actuation of the satellite to ensure a high level of accuracy when tracking debris. The future of this satellite can be expanded upon by figuring out way to implement accurate controllers such as the LQR controller, as well as improving the camera system that can track smaller pieces of space debris.

References

[1] Hall, L., “The History of Space Debris,” Space Traffic Management Conference, Nov 2014, pp. 3, retrieved July 2021.
https://commons.erau.edu/cgi/viewcontent.cgi?article=1000&context=stm

[2] Kan, S., “China’s Anti-Satellite Weapon Test,” Congressional Research Service, April 2007, retrieved October 2021.
https://apps.dtic.mil/sti/pdfs/ADA468025.pdf

[3] Gregory, F., “NASA Safety Standard: Guidelines and Assessment Procedures for Limiting Orbital Debris,” Office of Safety and Mission Assurance, August 1995, retrieved July 2021.
https://ntrs.nasa.gov/citations/19960020946

[4] Federal Communication Commission, “Mitigation of Orbital Debris in the New Space Age,” Federal Communication Commission, April 2020, retrieved August 2021.
https://docs.fcc.gov/public/attachments/DOC-363486A1.pdf

[5] Wie, B., “Space Mission Analysis and Design by Wiley Larson and James Wertz,” American Institute of Aeronautics and Astrodynamics, 2008.

[6] Dunbar, B., “Space Debris and Human Spacecraft,” National Aeronautics and Space Administrations, May 2021, retrieved July 2021.
https://www.nasa.gov/mission_pages/station/news/orbital_debris.html

[7] Kalinski, M. E., “Hypervelocity impact analysis of International Space Station Whipple and Enhanced Stuffed Whipple Shields,” Institution Archive of the Naval Postgraduate School, Dec 2004, pp. 7-11, retrieved October 2021.
https://calhoun.nps.edu/bitstream/handle/10945/1233/04Dec_Kalinski.pdf?sequence=1&isAllowed=y

[8] Japanese Aerospace Exploration Agency, “Commercial Removal of Debris Demonstration (CRD2),” Japanese Aerospace Exploration Agency, retrieved October 2021.
https://www.kenkai.jaxa.jp/eng/research/crd2/crd2.html

[9] Shan, M., Guo, J., Gill, E., “Review and Comparison of Active Space Debris Capturing and Removal Methods,” Progress in Aerospace Sciences, Vol. 80, pp.18-32, January 2016, retrieved October 2021. https://www.sciencedirect.com/science/article/pii/S0376042115300221

[10] Shuangyan S., Xing J., Chang H. “Cleaning space debris with a space-based laser system”. Chinese Journal of Aeronautics, Vol. 27, No. 4, pp.805-811, 2014, retrieved October 2021.
https://www.sciencedirect.com/science/article/pii/S1000936114001010#b0005

[11] Chapline, G., Rodriguez, A., Snapp, C., Dorsey, G., Fowler, M., Greene, B., Schneider, W., Scott, C., Pessin, M., Butler, J., Sparks, J.S., Bauer, P., Steinetz, B., Stevenson, C., "Thermal Protection System," National Aeronautics and Space Administration, retrieved October 2021. https://www.nasa.gov/centers/johnson/pdf/584728main_Wings-ch4b-pgs182-199.pdf

[12] Long, F., "Design and Testing of a Nanosatellite Simulator Reaction Wheel Attitude Control System," Utah State University, Logan, UT, 2014, retrieved August 2021. https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1456&context=gradreports

[13] "DC Motor Speed: System Modeling," Control Tutorials for MATLAB & Simulink, retrieved February 2022. https://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling

[14] Gaeid, K., "Optimal Gain Kalman Filter Design with Dc Motor Speed Controlled Parameters," Journal of Asian Scientific Research, 2013, 3(12):1157-1172, retrieved August 2021. https://archive.aessweb.com/index.php/5003/article/download/3582/5679

[15] Gajamohan, M., Muehlebach, M., Widmer, T., D'Andrea, R., "The Cubli: A Reaction Wheel Based 3D Inverted Pendulum," Universidade de Sao Paulo, Universidade Publica Brasil, San Paulo, Brazil, retrieved October 2021. http://www.usp.br/ldsv/wp-content/uploads/2017/11/Cubli_ECC2013.pdf

[16] Tzinis, I., "Technology Readiness Level," National Aeronautics and Space Administration, Oct 2012, retrieved August 2021. https://www.nasa.gov/directorates/heo/scan/engineering/technology/technology_readiness_level

[17] "3420 Dual Ball Bearing Long Life DC Motor," Handson Technology, retrieved September 2021. https://handsontec.com/dataspecs/motor_fan/XD3420-Motor.pdf

[18] "L298 DUAL FULL-BRIDGE DRIVER," STMicroelectronics (2000), retrieved September 2021. https://www.st.com/resource/en/datasheet/cd00000240.pdf

[19] "Toshiba Bi-CD Integrated Circuit Silicon Monolithic T B 6 6 1 2 F N G Driver IC for Dual DC motor," Toshiba (2007), retrieved September 2021. https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf

[20] Fischell, R., "Gravity Gradient Stabilization," APL Technical Digest, May 1964, retrieved February 2022. https://www.jhuapl.edu/Content/techdigest/pdf/APL-V03-N05/APL-03-05-Fischell.pdf

[21] Maurer, R., Fraeman, M., Martin, M., Roth, D., “Harsh Environments: Space Radiation Environment, Effects, and Mitigation,” Johns Hopkins APL Technical Digest, Vol. 28, No. 1, 2008, pp.17-28, retrieved October 2021.
https://www.jhuapl.edu/Content/techdigest/pdf/V28-N01/28-01-Maurer.pdf

[22] Rowberg, J., “i2cdevlib,” Github, retrieved March 2022.
https://github.com/jrowberg/i2cdevlib/tree/master/Arduino

[23] “Arduino Library and API,” PIXY Documentation, Retrieved July 2021.
https://docs.pixycam.com/wiki/doku.php?id=wiki:v2:arduino_api#arduino-library-and-api

[24] Beauregard, B., “Improving the Beginner’s PID – Introduction,” Brett Beauregard Project Blog, retrieved February 2022.
http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/

[25] Luuk, I., “Logging Arduino Serial Output to CVS/Excel (Windows/Mac/Linux),” Circuit Journal, retrieved March 2022.
https://circuitjournal.com/arduino-serial-to-spreadsheet

Appendices

Appendix A - Derivation of Transfer Functions

The transfer function of a brushed DC motor combines the Kirchhoff's Voltage Law equation with the physical impedance of the motor and the torque/current relationship. Below are the equations involved.

$$\tau = k_t \cdot I_a \tag{A.1}$$

$$\tau = J \cdot \dot{\omega} + b \cdot \omega \tag{A.2}$$

$$E_b = k_e \cdot \omega \tag{A.3}$$

$$V_a = I_a \cdot R + L_a \cdot I_a + E_b \tag{A.4}$$

Taking the Laplace Transform of these equations allow for them to be solvable in the s-domain.

$$\tau(s) = k_t \cdot I_a(s) \tag{A.5}$$

$$\tau(s) = J \cdot s \cdot \omega(s) + b \cdot \omega(s) \tag{A.6}$$

$$E_b(s) = k_e \cdot \omega(s) \tag{A.7}$$

$$V_a(s) = I_a(s) \cdot R + L_a \cdot s \cdot I_a(s) + E_b(s) \tag{A.8}$$

The torque equations can then be combined, and the current can be isolated.

$$k_t \cdot I_a(s) = J \cdot s \cdot \omega(s) + b \cdot \omega(s)$$

$$I_a(s) = \frac{J \cdot s \cdot \omega(s) + b \cdot \omega(s)}{k_t} \tag{A.9}$$

The current equation above as well as the back-EMF equation $E_b(s)$ can be used in the Kirchhoff's Voltage Law as shown below.

$$V_a(s) = \frac{J \cdot s \cdot \omega(s) + b \cdot \omega(s)}{k_t} \cdot R + L_a \cdot s \cdot \frac{J \cdot s \cdot \omega(s) + b \cdot \omega(s)}{k_t} + k_e \cdot \omega(s)$$

$$V_a(s) = \frac{J \cdot s + b}{k_t} \cdot \omega(s) \cdot (R + L_a \cdot s) + k_e \cdot \omega(s)$$

$$V_a(s) = \frac{(J \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t}{k_t} \cdot \omega(s) \tag{A.10}$$

As stated in the literature review, $k_e = k_t$. As such, the transfer function from the input voltage into a motor to the angular velocity outputted by the reaction wheel is given by the following transfer function.

$$\frac{\omega(s)}{V_a(s)} = \frac{k_t}{k_t(J \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t} \qquad (A.11)$$

The relationship between the angular velocity of the reaction wheel and the platform can be explained by the conservation of angular momentum.

$$J_{wheel} \times \omega_{wheel} = J_{system} \times \omega_{system}$$

$$\omega_{system} = \frac{J_{wheel}}{J_{system}} \times \omega_{wheel} \qquad (A.12)$$

The angular velocity of the satellite is given by the following transfer function

$$\frac{\omega_{system}(s)}{V_a(s)} = \frac{J_{wheel}}{J_{system}} \times \frac{k_t}{(J_{wheel} \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t} \qquad (A.13)$$

The format of the transfer function in the equation above is only able to output the angular velocity but can be modified to output angular displacement by replacing it with the derivative of the angular displacement. Since the transfer function is already in the s-domain, the Laplace transform of the derivative of the angular displacement can replace the angular velocity in the transfer function. The s variable can then be transferred to the right-hand side, which increases the rank of the transfer function from two to three, meaning that the transfer function will output the signal current, angular velocity and angular displacement.

$$\omega_{system} = \frac{d}{dt}\theta \qquad (A.14)$$

$$\omega_{system}(s) = \theta(s) \cdot s$$

$$\frac{\theta(s) \cdot s}{V_a(s)} = \frac{J_{wheel}}{J_{system}} \times \frac{k_t}{(J_{wheel} \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t}$$

$$\frac{\theta(s)}{V_a(s)} = \frac{J_{wheel}}{J_{system}} \times \frac{k_t}{((J_{wheel} \cdot s + b)(R + L_a \cdot s) + k_e \cdot k_t) \cdot s} \qquad (A.15)$$

Appendix B - MATLAB Code

```
%Transfer Function of one DOF reaction wheel system
%w_system/V_a = J_wheel/J_system * k_t/((J_wheel*s+b)*(R+L_a*s)+k_t*k_e)

clear all
close all
clc

J_wheel = 0.00331; %kg*m^2 Moment of Inertia of Wheel plus Wheel
J_system = 0.01836; %kg*m^2 Moment of Inertia of System
J_shaft = 0.000084; %kg*m^2 Moment of Inertia of Shaft

k_t = 0.49; %N*m/A Motor Torque Constant
k_e = k_t; %Back EMF Constant

b_v = 0.0000021; %Viscous Friction 0.0000021
r = 2.3; %Ohm Motor Resistance
L_a = 0.001339; %Henry Motor Inductance
V_a = 12; %Volts Applied Voltage

b = [0 0 0 J_wheel*k_t];%*V_a
a = [J_wheel*J_system*L_a J_system*(J_wheel*r+L_a*b_v) J_system*(k_e*k_t+b_v*r)
0];
%{
%incorrect Transfer function
b = [0 0 J_wheel*k_t];%*V_a
a = [J_wheel*J_system*L_a J_system*(J_wheel*r+L_a*b_v) J_system*(k_e*k_t+b_v*r)];
%}

[A,B,C,D] = tf2ss(b,a)

Co = [B A*B (A^2)*B] %Controllability Matrix
Ob = [C;
      C*A;
      C*(A^2)] %Observability Matrix

det(Co)
det(Ob)

E = eig(A)

G = tf(b,a)
GPole = pole(G)

sys = ss(A,B,C,D);
step(sys)

%State-feedback controller design
%tf = 0.1;
tf = 16.3;
dt = 1E-2;

%Using LQR Control Theory
```

```
%define Q and R Matrices
%start with identity matrix and get respeonse as fast as possible
Q = [1 0 0;
    0 1 0;
    0 0 100000000000];
% Q = (C'*C);
R = 1;

%N = 1; %Error Compensation
N = 15.867; %Error Compensation

K_lqr = lqr(A,B,Q,R)

open_system("PIDvsLQR.slx");
sim("PIDvsLQR.slx");

for i = 1:numel(ans.tout)
    desiredpoint(i) = 1;
end

%{
figure(1)
plot(ans.tout,ans.simout(:,1));
hold on
plot(ans.tout,ans.simout(:,2));
xlabel('Time (sec)');
ylabel('Displacement (degrees)');
title('Incorrect Transfer Function LQR Response');
legend('Reference Point','LQR Response')
%}

figure(2)
plot(ans.tout,ans.simout1(:,1));
hold on
plot(ans.tout,ans.simout1(:,2));
hold on
plot(ans.tout,ans.simout1(:,3));
xlabel('Time (sec)');
ylabel('Magnitude');
title('PID vs LQR Controller');
legend('Reference Point','PID Controller','LQR Controller')
% legend('Reference Point','LQR Controller')
%}


%test run 6
%11 tests done
%roughly 17 seconds

%Physical experiment using IMU sensor to measure the rotational position of the
satellite that's tracking a ball at 10 degrees from
%initial starting point
ExperimentalData = xlsread('testgodknows.xlsx');
t=(0:(16.3/503):16.3)';
TestAvg = mean(ExperimentalData(:,1:5),2);
```

```
figure(3)
plot(t,ExperimentalData(:,1));
hold on
plot(t,ExperimentalData(:,2));
hold on
plot(t,ExperimentalData(:,3));
hold on
plot(t,ExperimentalData(:,4));
hold on
plot(t,ExperimentalData(:,5));
hold on
plot(t,ExperimentalData(:,6));
hold on
plot(t,ExperimentalData(:,7));
hold on
plot(t,ExperimentalData(:,8));
hold on
plot(t,ExperimentalData(:,9));
hold on
plot(t,ExperimentalData(:,10));
hold on
plot(t,ExperimentalData(:,11));
hold on
plot(ans.tout,ans.simout1(:,2));
xlabel('Time (sec)');
ylabel('Magnitude (deg)');
title('Experimental vs Simulated PID Tracking');
legend('Test Run 1','Test Run 2','Test Run 3','Test Run 4','Test Run 5','Test Run
6','Test Run 7','Test Run 8','Test Run 9','Test Run 10','Test Run 11','Simulated
PID')

figure(4)
plot(t,TestAvg);
hold on
plot(ans.tout,ans.simout1(:,2));
xlabel('Time (sec)');
ylabel('Magnitude (deg)');
title('Average Experimental vs Simulated PID');
legend('Experimental Average','Simulated PID')

figure(5)
plot(t,ExperimentalData(:,1));
hold on
plot(ans.tout,ans.simout1(:,2));
xlabel('Time (sec)');
ylabel('Magnitude (deg)');
title('Experimental vs Simulated PID Tracking');
legend('Test Run 1','Simulated PID')
```

Appendix C - Arduino Code

```
#include <Pixy2.h>
#include <PID_v1.h>

//refer to the MPU6050_DMP6 example for how it works
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
   #include "Wire.h"
#endif
MPU6050 mpu;

#define OUTPUT_READABLE_YAWPITCHROLL

#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
   mpuInterrupt = true;
}
```

```
// This is the main Pixy object
Pixy2 pixy;

//variables that we declare
int signature, x, cx, width, cx0;

//motor pin polarity
int out_B_PWM = 13;
int out_B_IN4 = 12;
int out_B_IN3 = 11;
int out_A_IN2 = 10;
int out_A_IN1 = 9;
int out_A_PWM = 8;
//area where object will not move
int deadZone = 32; //10% of 320

double Setpoint; //desired point
double Input; //current position
double Output; //motor speed


//PID VALUES THAT NEED TO BE ADJUSTED
double Kp = 15.14, Ki = 14.92, Kd = 0.95;
//double Kp = 0.53, Ki = 0.01, Kd = 2;

//creates PID instance
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);

void setup()
{
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation
difficulties
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
  #endif

  Serial.begin(115200);
  while (!Serial);
  Serial.println(F("Initializing I2C devices..."));
  mpu.initialize();
  pinMode(INTERRUPT_PIN, INPUT);

  Serial.println(F("Testing device connections..."));
  Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050
connection failed"));

delay(10000);

  // load and configure the DMP
  Serial.println(F("Initializing DMP..."));
  devStatus = mpu.dmpInitialize();

  // supply your own gyro offsets here, scaled for min sensitivity
  mpu.setXGyroOffset(43);
  mpu.setYGyroOffset(19);
  mpu.setZGyroOffset(5);
  mpu.setZAccelOffset(989); // 1688 factory default for my test chip

  // make sure it worked (returns 0 if so)
  if (devStatus == 0) {
    // Calibration Time: generate offsets and calibrate our MPU6050
    mpu.CalibrateAccel(6);
    mpu.CalibrateGyro(6);
    mpu.PrintActiveOffsets();
    // turn on the DMP, now that it's ready
    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    // enable Arduino interrupt detection
    Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
    Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
    Serial.println(F(")..."));
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();

    // set our DMP Ready flag so the main loop() function knows it's okay to use it
    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady = true;

    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
  }

  // configure LED for output
  pinMode(LED_PIN, OUTPUT);

    //if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer))
    //{ // Get the Latest packet
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
            Serial.print("\n");
    //}
    delay(3000);

  Setpoint = 0; //camera will try to center on the ball

  pixy.init(); //activates the pixy cam
  myPID.SetMode(AUTOMATIC); //activates PID

  myPID.SetTunings(Kp,Ki,Kd);

  //sending commands to the motor driver
  pinMode(out_B_PWM,OUTPUT);
  pinMode(out_B_IN4,OUTPUT);
  pinMode(out_B_IN3,OUTPUT);
  pinMode(out_A_IN2,OUTPUT);
  pinMode(out_A_IN1,OUTPUT);
  pinMode(out_A_PWM,OUTPUT);
}

void loop()
{
    // if programming failed, don't try to do anything
    if (!dmpReady) return;
    // read a packet from FIFO
    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer))
    { // Get the Latest packet
        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
```

```
          mpu.dmpGetGravity(&gravity, &q);
          mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
          //Serial.print("yaw\t");
          //Serial.print("pitch\t");
          //Serial.print("roll\n");
          Serial.print(ypr[0] * 180/M_PI);
          Serial.print("\t");
          Serial.print(ypr[1] * 180/M_PI);
          Serial.print("\t");
          Serial.println(ypr[2] * 180/M_PI);
          Serial.print("\n");
        #endif



        pixy.ccc.getBlocks();
        if (!pixy.ccc.numBlocks)
        {
          digitalWrite(out_B_IN4,LOW);
          digitalWrite(out_B_IN3,LOW);
          analogWrite(out_B_PWM,0);
        }
        if (pixy.ccc.numBlocks)
        {
          Input = pixyCheck();
          //Serial.print(Input);
          if (Input >= -deadZone && Input <= deadZone)
          {
            digitalWrite(out_B_IN4,LOW);
            digitalWrite(out_B_IN3,LOW);
            analogWrite(out_B_PWM,0);

            //Serial.print(" Center ");
            //Serial.print('\n');
          }
          else if (Input > 0)
          {
            //determines polarity of motor
            digitalWrite(out_B_IN4,LOW);
            digitalWrite(out_B_IN3,HIGH);

            Input = -Input; //the PID isnt able to output a negative analog value because the analog
range is from 0 to 255
            //this flips the value so the PID is able to output a positive analog value
            //see PID Bret's equation on his website
http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/
```

```
            //PID caluclate the nessecary speed
            myPID.Compute();

            analogWrite(out_B_PWM,Output);

            //Serial.print(" Spin Right ");
            //Serial.print(Output);
            //Serial.print('\n');
          }
          else if (Input < 0)
          {
            digitalWrite(out_B_IN4,HIGH);
            digitalWrite(out_B_IN3,LOW);

            myPID.Compute();

            analogWrite(out_B_PWM,Output);

            //Serial.print(" Spin Left ");
            //Serial.print(Output);
            //Serial.print('\n');
          }
        }
  }
delay(10);
}

double pixyCheck()
{
  int i;
  // grab blocks!
  pixy.ccc.getBlocks();

  // If there are detected blocks, get info
  if (pixy.ccc.numBlocks)
  {
    for (i=0; i<pixy.ccc.numBlocks; i++)
    {
      //position of bottom left corner of the box
      x = pixy.ccc.blocks[i].m_x;

      //width of the box
      width = pixy.ccc.blocks[i].m_width;

      //center of the box
      //cx = (x + (width / 2));

      //this shrinks the range from -1 to 1 and works with the deadzone
      //cx0 = map(cx, 0, 320, -160, 160);
      cx0 = map(x, 0, 320, -160, 160);
    }
    return cx0;
  }
}
```