Project ToppleBot

A Scalable Cube Robot for Autonomous Self-Balancing

Kilian Olen
Controls and Manufacturing
Embry-Riddle Aeronautical University
Daytona Beach, Florida
olenk@my.erau.edu

William Baker
Electrical and Systems
Embry-Riddle Aeronautical University
Daytona Beach, Florida
bakerw7@my.erau.edu

Gabriel Rodriguez
Software and Communication
Embry-Riddle Aeronautical University
Daytona Beach, Florida
rodrig43@my.erau.edu

Erin Bader
Mechanical Design
Embry-Riddle Aeronautical University
Daytona Beach, Florida
badere@my.erau.edu

Submitted to the Engineering Physics Review Board in partial
fulfillment of the requirements for EP 496: Space Systems Design I

at the

EMBRY-RIDDLE AERONAUTICAL UNIVERSITY

Principal Investigator / Project Lead……...……..
November 4, 2024

Accepted by……...
Aroh Barjatya
Professor of Engineering Physics
Director, Space and Atmospheric Instrumentation Lab

Abstract

This paper presents the Conceptual Design Review for Project ToppleBot, whose mission is to design, manufacture, and control a scalable wheel cube rover. The cube's design must be transferable to variable cube sizes, providing a procedure for others to replicate and iterate over different configurations for their specific payload requirements. Applications of such a rover are plentiful, mitigating control dilemmas that face other robotic platforms in space applications and improving reproducibility and scalability through its efficient, straightforward design. ToppleBot will build off the successes of CUBEBAS [1], REM-RC [13], Cubli [2] and more by developing a similar design with improvements to the system architecture through the integration of additional communication abilities, structural improvements, electrical systems, control behaviors, and a novel sizing algorithm.

Utilizing the capabilities of micro-ROS, ToppleBot will communicate wirelessly with the control station allowing for direct, real-time visualization of the cube's orientation and real-time control over the cube's high-level behaviors, such as balancing, spin, kip-up, and topple. Such a framework also allows for the control of a multi-agent system. The microprocessor onboard the cube will receive the high-level behavior commands and transfer them to low-level motor commands through a closed-loop control algorithm integrated with an onboard IMU. The fully integrated electrical system, ensuring safe and precise operation will ensure that the low-level commands are delivered to the respective motors. Structurally, the ToppleBot will feature the integration of threaded inserts for increased stability, additive manufacturing materials for rapid prototyping (PLA and PETG), and damping mechanisms to minimize structural loads in toppling behaviors. Lastly, using the parameters derived from mathematical and CAD models, an algorithm will be developed such that a user may input a desired ToppleBot dimension, and will receive the necessary parameters for its creation and deployment.

The proposed ToppleBot project is to be completed in partial fulfillment of the requirements for EP 496, Space Systems Design I. It will also provide Embry-Riddle's Physical Sciences Department a scalable architecture for future senior design robotics projects, with an accompanying sizing algorithm and additional software.

Table of Contents

List of Figures

List of Tables

1. Introduction

Self-balancing cubes, such as those developed in projects like CUBEBAS and ETH Zurich's Cubli, utilize reaction wheels to maintain active stability along a point or edge. These systems operate on similar principles to the inverted pendulum problem; however, their ability to move in all six degrees of freedom (DOF), including rotational and translational motion, provides a compelling platform for studying balance and stability. Unfortunately, most previous designs have relied on external intervention to bring the cube into a balancing pose. Project ToppleBot aims to address this limitation by incorporating a braking mechanism, allowing the cube to initiate its own balancing pose, as Cubli demonstrated.

Building on the foundations of CUBEBAS and Rem-RC, ToppleBot seeks to implement significant advancements in system control, operation, and scalability. In addition to the braking mechanism, ToppleBot will refine control algorithms, enhance communication capabilities, and improve structural design, making the platform more adaptable to different cube sizes. These improvements will allow for greater stability and user customization, providing a system that can be easily replicated for future applications.

Even in the event of a braking mechanism failure, ToppleBot will be designed to implement considerable advancements in control precision and stability compared to CUBEBAS. The project focuses on refining movement, balance, and overall system reliability, ensuring better performance in dynamic environments. By building upon prior work and addressing key limitations, ToppleBot seeks to improve the self-balancing cube design workflow, producing comparable functionality to Cubli with a scalable framework for future developments.

2. Systems Overview

2.1. Mission Statement

To design and manufacture a scalable reaction wheel cube robot capable of balancing on its edges and corners. The ToppleBot will demonstrate capabilities across different sizes, with an accompanying algorithm to facilitate its adaptable creation across different sizes, payloads, and applications.

2.2. Mission Objectives

2.2.1. Primary Mission Objective

To achieve balancing on corners and edges, resistant to disturbances using a closed-loop control algorithm while displaying the cube's orientation through the control station.

2.2.2. Secondary Mission Objective

To develop a scalable algorithm that allows future users to build a ToppleBot of arbitrary size and payload, adjusting for inertial, structural, and control differences across sizes.

2.2.3. Tertiary Mission Objective

To achieve locomotion control and odometry tracking through the use of balance, spin, kip-up, and topple behaviors with sensor integration.

2.3. Systems Requirements

2.3.1. Functional Requirements

The ToppleBot must be capable of balancing on both its edges and corners, with real-time orientation data transmitted to the command station. Its mathematical and structural design should be flexible enough to generate unique configurations based on random inputs. Additionally, the ToppleBot should execute controlled locomotion, rotation, and kip-ups, all while continuously sending performance data back to the control station.

2.3.2. Operational Requirements

The ToppleBot must achieve specific operational requirements for the project to be successful. It must operate continuously for a minimum of 30 minutes. Additionally, it should withstand a drop of 0.5 meters while remaining structurally secure and be capable of enduring moderately harsh environments. From a communication and control perspective, the ToppleBot must maintain communication with the command center from at least 10 meters away, with an angular deviation of less than one degree when balancing. Furthermore, the CAD model must support scalability within 5% to ensure future adaptability.

Table 1: Mission Requirements		
Functional	Primary Mission Performance	Maintain balance on edges
		Maintain balance on corners
		Send orientation data to the control station
	Secondary Mission Performance	Provide design parameters such as torque requirements, power requirements, and structural dimensions, based on overall dimension inputs.
	Tertiary Mission Performance	Perform spin, kip-up, and topple behaviors.
		Perform locomotion based on control station commands.
		Maintain stability and odometry tracking throughout locomotion.
Operational	Operational Duration	Maintain 30 minutes of continuous operation.
	Resilience	Withstand drop from 0.5m height
	Reliability	Physical barrier from outside elements
	Communication	10+ meters indoors
	Control	Maintain angular deviation within 1 degree
	Scalability	Within a 5% of CAD model

2.4. Constraints

As with any design project, there are various constraints related to the design of the ToppleBot. These constraints are detailed below, in Table 2. It is important to note that this project is to be fully completed within 8 months; thus, quick action is necessitated from the team to get the system operational and begin testing and optimization. Another notable constraint is the manufacturing constraint of 3D printing bed area. Due to this limitation, larger cubes will be harder to test, meaning that the sizing algorithm will have less accurate results for larger systems.

Table 2: Mission Constraints		
Constraints	Cost	Limited to a ~ $1,000 budget for all components, manufacturing, and materials
	Operating Date	Must be fully operational by May 1, 2025
	Operating Location	Primarily designed for indoor use, and space applications cannot be tested
	Control	Must utilize momentum wheels and brakes only
	Manufacturing	Printing is limited to the 3D printing bed area and materials
	Communication	Communication must be supported by micro-ROS

2.5. System Context Diagram

Figure 1 depicts the framework of ToppleBot's operation. Briefly summarizing the information shown, the onboard IMU and microprocessor will take the ToppleBot's current state and desired behaviors (sent from the control station), and convert them into low-level motor commands. This conversion will be facilitated by an onboard control algorithm, with all computation housed onboard the ToppleBot for minimal control latency and maximum control authority. ToppleBot's state will also be broadcasted to the control station such that it can be displayed through RVIZ. In order to receive the information, the control station will need to deploy a micro-ROS agent, bridging the gap between Micro-ROS and ROS2 Humble onboard the control station.

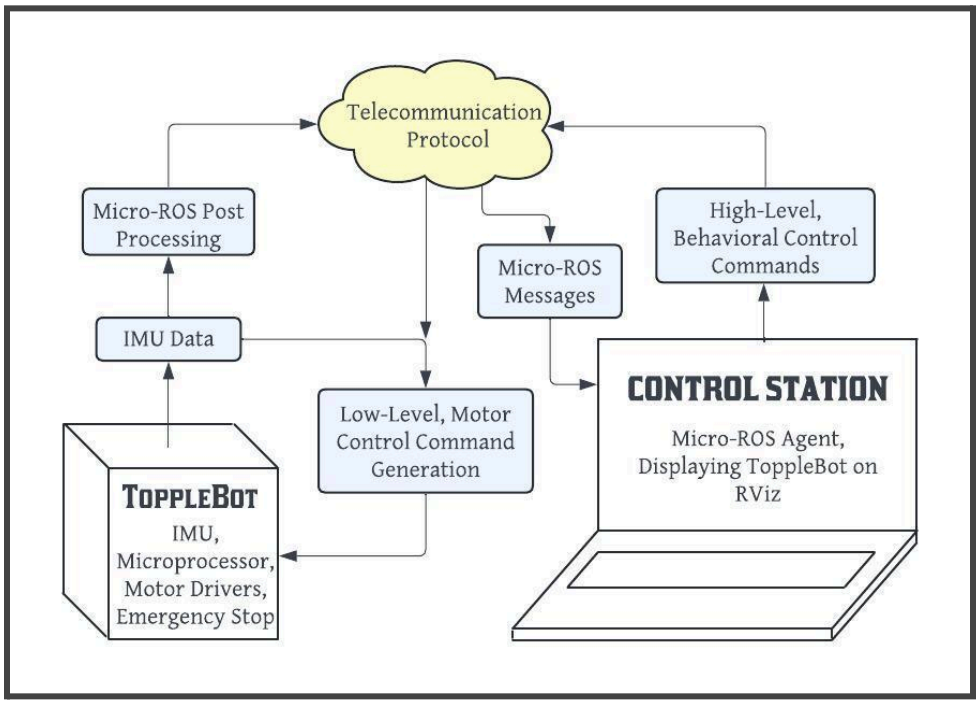

Figure 1: ToppleBot System Context Diagram

3. Structural System

ToppleBot's structural design must fulfill the requirements in Table 3 to have a suitable structure for its function. Drawing from the initial design of the CUBEBAS project, additional changes to the structure must be implemented to account for the new objectives of ToppleBot. New motions such as the "kip-up" or "topple" behavior emphasize the importance of structural resilience. Incorporating a breaking mechanism to accomplish this will also call for a reorganization of the motor, wheel, and electrical mounting that differs from the system's original design.

To execute the established requirements considerations of materials, reaction wheel, motor mounting, braking mechanism, and microcontroller mounting will be explored. These changes aim to successfully fulfill the structural requirements of the project to create a durable, scalable, and controllable system.

Table 3: Structural System Requirements									
Number	**Requirement**	**Measure of Performance**		**Justification**	**Owner**	**Validation**	**Priority**	**Status**	**Remarks**
		Threshold	**Objective**						
3.1	The cube rover must withstand repeated shocks to its frame.	Shock from Rotation	Shock from Controlled Drop (0.5 meters)	The locomotion of the cube rover produces a consistent impact as the system transitions between statically stable positions.	Erin	FEA Models and Physical Testing	1	Not Started	
3.2	The reaction wheels must provide sufficient control authority.	Balancing Capabilities	Kip-Up Capability	Necessary to perform any locomotion or "kip-up" behaviors.	Erin	Mathematical Models and Physical Testing	1	Not Started	
3.3	Each reaction wheel must be equipped with a durable braking mechanism.	100+ Cycles	1000+ Cycles	Improves system practicality by reducing component replacements.	Erin	Isolated System Endurance Testing	1	In Progress	
3.4	The reaction wheel mounts must withstand abrupt spikes in reaction torques.	Sufficient torque to produce Kip-Up reaction.	Sufficient torque to stop motors at full speed.	The system must be structurally sound.	Erin	FEA Models and Physical Testing	1	Not Started	Required torques will be determined with CAD.
3.5	The design of the cube rover should be scalable with defined parameters.	Parametric Design	N/A	Reduces the need to completely remodel components should parameters change.	Erin	CAD	2	In Progress	
3.6	A sizing algorithm (Python or MATLAB) shall produce system requirements and specifications from the input cube size.	Output of Basic Parameters (Torque, Power, etc.)	Output of Comprehensive Parameters	Allows for easier replication by similar project groups.	Erin	Additional ToppleBot Iteration Sourced from Algorithm	2	Not Started	

3.1. Material Considerations

ToppleBot will be experiencing structural impacts during motion, high torques, and rapid braking. Therefore, the materials used for printing must be able to sustain shocks to the frame and be durable enough not to deform from repeated testing. Currently, materials under consideration for the structural aspects of ToppleBot, specifically the framing and mounting, are PETG and PLA. PLA would prove useful for initial prototypes of this project because it tends to be a more workable material and easier to print with. This would be a reasonable choice because it allows for cheaper and simpler printing in cases where reprinting for the sake of refinement may be common. PETG, conversely, would be a more viable option for a finalized design in which frequent testing will be present. PETG does have an increased price and is more difficult to print with due to its higher melting point. Yet, it will be rewarded with higher impact and abrasion resistance, which will decrease the likelihood of fractures during testing; therefore, extending the longevity of the structure. Overall, the cost and printing differences between PLA and PETG would be considered minute, making PETG the most likely material option for the main structure of ToppleBot across all portions of the design process. TPU is currently being considered for the implementation of corner caps with the purpose of shock absorption. The elastic and rubbery properties of TPU will increase the longevity of the frame during repeated impacts during motion. For future iterations, constructing the frame from aluminum would increase the durability of the frame and mounting components. This is currently not viable during the prototyping stage of development.

3.2. Reaction Wheel Options

Two design options for the reaction wheel are viable for the ToppleBot. The first of which would be based on the original CUBEBAS wheel design that was 3D printed. For this design, the printed wheels were weighted using nuts and bolts around the perimeter. This method of weighting the wheel would make the wheel more textured and less viable for the mounting and breaking options to be discussed below. This method of wheel construction would be cheaper and more accessible because it can be printed in the lab. The second design option for the wheel would be machining it using aluminum. This would allow for the weighting to be distributed throughout the wheel, and it would allow for smoother braking and structural organization. Unfortunately, this option is both pricier and less accessible compared to the previous consideration.

3.3. Motor Mounting Options

Motors for the wheels would be mounted using an offset structure that would be directly connected to the internal side of the ToppleBot. This structure would be 3D-printed and will create a gap for the wheel to sit internally. The mounting structure will be placed on three of the sides allowing for each of the three wheels to have sufficient room internally and create a space for the braking system to interact with the wheel. Other mounting considerations would be a main center mount where each other the three motors will be placed. This would allow for a greater space for the implementation of a braking mechanism which, depending on the braking mechanism decision, would be necessary. Since the current path of the braking mechanism is the brake pad, the center-mounted motors would be the most viable choice to maximize the amount of space allowed for the internal wheels and brakes.

3.4. Braking Mechanism Options

A breaking mechanism would be incorporated with the wheel to abruptly halt the wheel's motion. This could be done using brake pads or a motorized pole. Brake pads would be a friction-based stopping method, this form of braking would have increased longevity compared to a collision-based braking method such as a motorized pole. The brake pad would be a curved plate attached to a motor, that would then be put into contact with the edge of the wheel to provide a friction force that opposes the wheel's motion. A motorized pole would be implemented as a "stick" or pole that is attached to a motor, when activated this pole would collide with the bolts, or other parts of the structure to provide instantaneous braking. Unfortunately, this method has been tested in similar projects and did not withstand repeated impact for extended testing. Therefore, with our current design plans, a brake pad would be the most probable consideration for a braking mechanism.

Table 4: Braking Mechanism Comparison

Braking Option	Advantages	Disadvantages
Brake Pad Systems	● higher durability which will equate to longevity ● Less damage to the wheel because of a lack of collision ● Higher surface area contact which would lead to more precise braking	● Larger mechanism, less space-efficient ● Possibly slower breaking, thus it may not be able to produce the required force
Pole	● More compact mechanism ● Faster braking due to immediate and forceful contact	● Collision braking would cause more damage to parts, thus decreasing the longevity of both the pole and wheel ● Less controlled breaking due to decreased contact area

3.5. Microcontroller and Sensor Mounting

In previous models of the CUBEBAS project, microcontroller mounting was placed in the center of the frame, along with the battery and other electronics. This method of mounting allowed for the center of mass to remain close to the center of the cube. With the implementation of a breaking mechanism, the microcontroller and other sensors could be placed internally on the non-wheel sides of the frame. This would allow the mass of the internal materials to be more evenly dispersed after the addition of the braking mechanism, as well as, allow for an increased space to be utilized for motor mounting structures. With the wheels being placed internally, as opposed to CUBEBAS where they were external, and the likelihood of a brake pad being implemented, side mounts for the electronics would allow for a better utilization of space and a balanced structure.

4. Electrical System

The electrical systems are critical to the functionality of ToppleBot, facilitating seamless communication between all components. There are four primary areas that the electrical system must address to ensure effective operation. As detailed in the table below, these sections function independently yet must synchronize to enable precise control from the control station.

Several challenges have been identified within the electrical system. A significant concern is power distribution and protection against reverse currents during motor braking. This challenge arises from the necessity for the chosen battery to supply adequate amperage for three motors, the microcontroller, and the IMU. Furthermore, the braking system may generate reverse currents, which pose a risk of damaging various components.

The circuit structure is straightforward, utilizing a PCB to ensure secure connections that can operate effectively in any orientation. The design incorporates a microcontroller, IMU, PCB relay, and motors equipped with integrated motor drivers. These components were selected based on their compatibility demonstrated in preliminary testing, facilitating the development of a functional prototype.

To ensure the safety of both the ToppleBot and the user, two main features are being added: diodes and a relay, which will protect specific circuit elements. Further details are provided in sections 4.3.1 and 4.2.2.

The image below illustrates the third iteration of the circuit diagram. The pinouts for the microcontroller to the various components are currently provisional; specific assignments will be finalized upon completion of the coding and the determination of final design choices. This layout represents the current arrangement of all necessary components for the circuit's functionality.

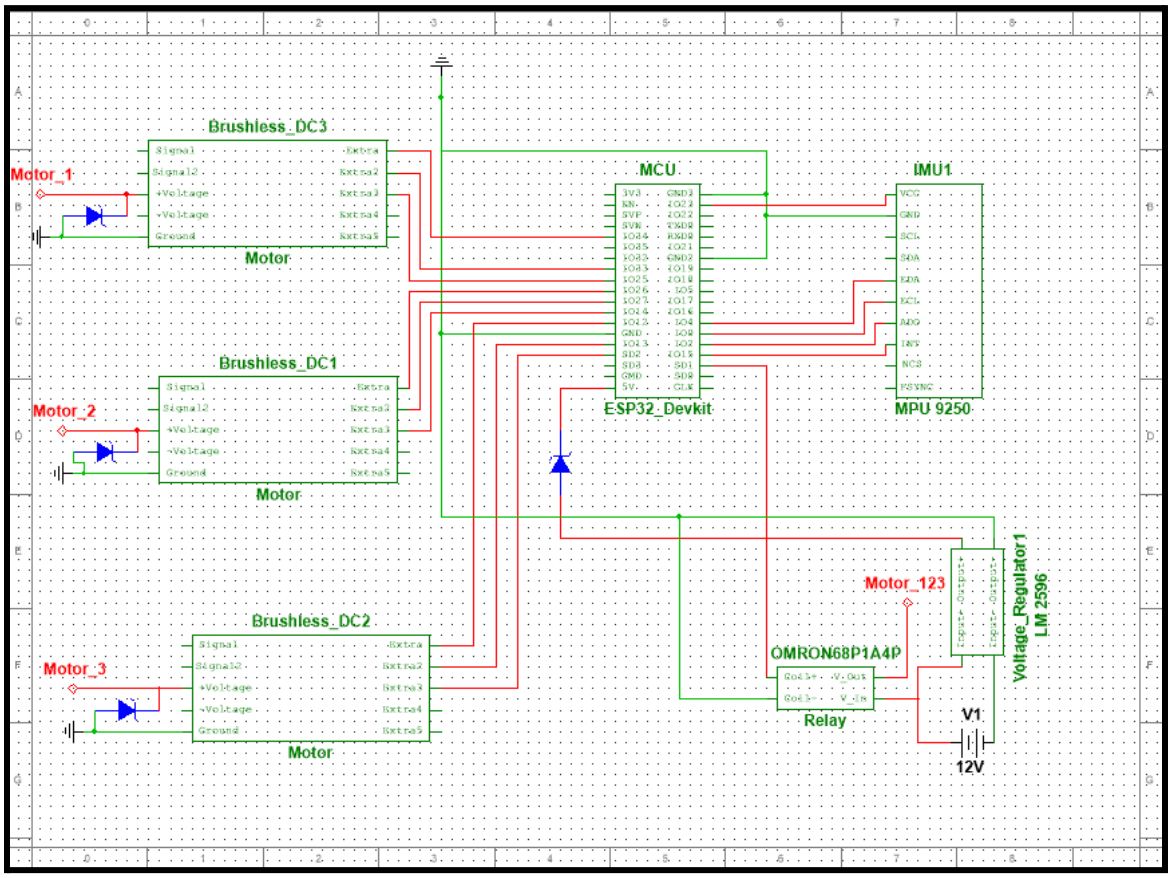

Figure 2: Preliminary Circuit Design Diagram.

Table 5: Electrical System Requirements									
Number	**Requirement**	**Measure of Performance**		**Justification**	**Owner**	**Validation**	**Priority**	**Status**	**Remarks**
		Threshold	**Objective**						
5.1	The cube rover must have a battery that adequately powers all subsystems.	10 minutes	30 minutes	DC motor performance scales with available power.	William	Circuit Design and Drain Cycle	1	In Progress	
5.2	The cube rover must possess an Emergency Stop mechanism that instantly cuts power to all subsystems.	N/A	N/A	Every electrical prototype with powerful actuators should have an E-Stop.	William	Circuit Design and Physical Testing	1	Not Started	
5.3	The cube rover must be able to control the motors adequately for precise control (MOTOR DRIVER etc.)	N/A	N/A	Necessary for proper functionality and gives the ability to topple with precision.	William	Testing to Ensure Motors Stop Quickly and Safely	1	In Progress	
5.4	Provide stable communication with all of the different parts to ensure the sensors run smoothly.	Stable Communication During Balancing	Stable Communication During Toppling	Having a strong connection will let the ToppleBot stay working for longer.	William	Testing of Systems and Continuous Toppling	1	Not Started	

4.1. Power Management Options

Power management is a critical aspect of the system design, requiring careful consideration of several factors. Adequate power must be supplied to all components, ensuring the appropriate voltage levels are maintained. A voltage divider is utilized for stepping down the voltage from the battery to the microcontroller. Additionally, protective measures must be implemented to safeguard against potential back currents generated by the motors during braking. These considerations are essential to ensure that all components operate harmoniously within ToppleBot's circuitry.

4.1.1. Battery Selection

The battery selection is crucial for a successful project. The battery must be powerful enough to operate the three motors and the microcontroller for an adequate amount of time. The current battery consideration is the GlobTek BL2600C1865003S1PGMG (BL2). This battery provides 11.1V and 2.6 Ah, which should provide enough runtime to properly test and demonstrate the ToppleBot. The BL2 is a rechargeable battery which is crucial for quick testing, reliability, and cost-effectiveness. Moreover, the wire connections provide an easy way to connect this battery to the terminals of the voltage divider and emergency relay. Finally, this battery is fairly small which is important to fit inside of the ToppleBot.

Figure 3: Picture of GlobTek Li-ion Battery.

4.1.2. Voltage Regulator

A voltage regulator is necessary since the 11.1V from the BL2 battery would break the microcontroller without one. An LM2596 voltage regulator will be utilized for this application. The voltage divider that is being used is built on a PCB board with built-in capacitors and resistors for fine adjustments of the output voltage that will be placed on our PCB board. The input of this voltage regulator is 3V to 40V and can be adjusted and limited to any voltage within

that range. With this regulator the battery will be plugged in and a voltage of exactly 5V will be output to power the microcontroller and any subsequent components.

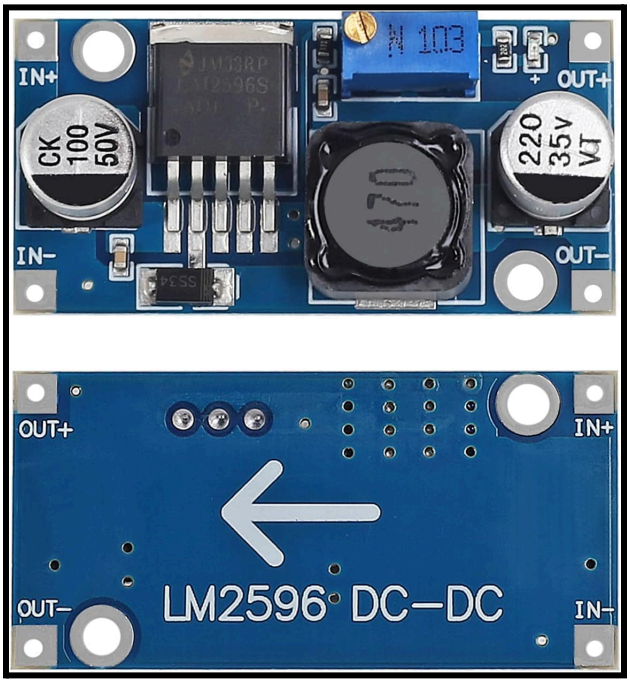

Figure 4: Picture of LM2596 Voltage Regulator.

4.1.3. Battery, Microcontroller, and Motor Protection

Each component requires protection, as the braking mechanism could generate reverse currents and voltages that may damage sensitive parts. To address this, a diode will be implemented to safeguard the components by allowing current to flow in only the desired direction. This protection is particularly important for the microcontroller, ensuring that reverse voltage does not reach and damage the microcontroller.

4.1.4. Thermal Considerations

The thermal restrictions are very lenient since the cube will not be sealed in for our preliminary full builds. The cube being open to airflow allows the battery, motors, and diodes to dissipate heat effectively and freely to be of little concern in the preliminary testing. On the other hand, in future applications when the ToppleBot is fully enclosed protecting the circuitry inside from harsh environments the thermal properties of each component and proper ventilation will be crucial considerations. Some thermal regulation methods under consideration include fans and ducts to control the temperature within the enclosed ToppleBot.

4.2. Emergency Stopping System

A new emergency-stopping system will be designed and implemented in the circuitry to ensure user and equipment safety. The function of the emergency stop will be to automatically disconnect the battery by using a killswitch within the circuitry. The GUI in the control station

will feature a prominent red button that sends a signal to the relay, disconnecting power to the motors and bringing them to a quick stop, preventing uncontrolled operation and potential hazards.

4.2.1. Relay Choice

We are selecting the OMRON G8P-1A4P relay, a straightforward relay that will connect to the microcontroller. Connecting the relay to the microcontroller is essential, as it allows control via the GUI. Upon receiving a signal from the microcontroller through the control station, the relay will cut all power to the motors. This Relay has an input voltage range of 10.2 to 13.2V so the voltage from the battery will work.

Figure 5: Picture of OMRON Electrical Relay.

4.2.2. Safety Feature

Implementing an emergency-stopping feature is essential for protecting both the ToppleBot and ensuring the safety of manual operation. If the motors become unresponsive or exhibit runaway behavior, the system allows us to cut power to the motors remotely from the control station, forcing an immediate shutdown. This override capability is crucial, as it ensures that the rapidly spinning motors can be stopped without manual intervention

4.3. Motor Control System

We are currently using a motor with an integrated control system, allowing the motor driver to be housed within the motor itself. This integrated design optimizes space and reduces power requirements, resulting in a simplified circuit layout that enhances organization and cleanliness. The built-in motor driver also facilitates more efficient control, enabling precise motor starting and stopping for balance and movement. For our preliminary tests, we are utilizing the Japan

24H Brushless Servo Motor, equipped with a Drive 100 Line Encoder and PWM speed regulation. This setup enables speed adjustments through a Pulse Width Modulation (PWM) signal sent directly to the built-in driver circuit.

Various motors are being evaluated to identify the optimal choice for each specific cube size, as torque requirements and related calculations will vary with size. For instance, the Cubli utilizes Maxon EC 45 Flat 50W BLDC motors [13], while the REM-RC uses Nidec 24H motors[14]. The motors are all very similar to each other yet there are at different quality levels that reflect on the specifications.

The integrated motor driver provides additional advantages, including streamlined control and minimized space requirements. This configuration allows for rapid motor response to control signals, reducing latency between command and action. Furthermore, an integrated driver-motor setup eliminates compatibility concerns, significantly reducing development and testing time and allowing for focus on other critical system considerations. In the future, we will continue to use motors with integrated motor drivers only with better specifications. The cohesive design enhances reliability, as the motor and driver are purpose-built to function seamlessly together.

Table 6: DC Motor Comparisons

Motor	Efficiency	Torque	Reliability	Price	Customization
Maxon EC-45	Very High	High	Very High	High	Extensive
Nidec 24H	High	Moderate	High	Moderate	Limited
Japan 24H	Moderate	Moderate to High	Moderate	Low	Moderate

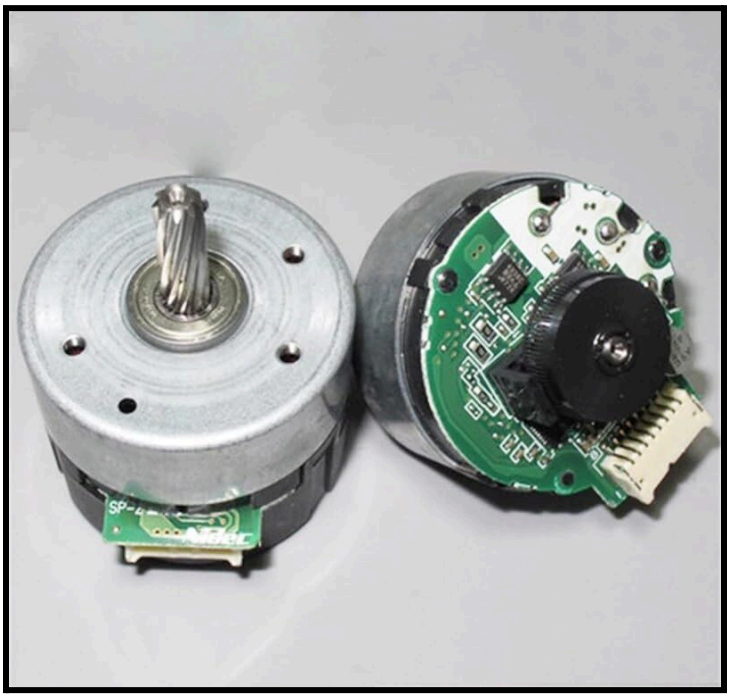

Figure 6: Picture of Japan 24H BLDC Motors.

4.4. Sensor Communication Protocol and Proper Connections

The system must operate effectively in any orientation, and proper grounding is essential to protect components from reverse voltage generated by the motors. All wiring must be securely contained within the structure, as the ToppleBot will rotate frequently, and space is limited due to the compact cubic design. A reliable communication protocol and secure connections are crucial for ToppleBot's functionality; the system cannot operate without stable connections. Various strategies are being implemented to ensure robust communication and secure connections throughout the device.

4.4.1. Communication

An Inter-Integrated Circuit (I2C) bus is a communication system that links multiple devices (such as sensors and peripherals) to a microcontroller using only two wires: SDA (Serial Data) and SCL (Serial Clock). These two wires allow all connected devices to share the same bus. Each device on the bus has a unique ID, so when the microcontroller sends data, it includes the target device's ID along with the command. This enables efficient communication across multiple devices with minimal wiring, as each device recognizes and responds only to commands that match its ID. This communication system still uses two other separate wires for power and ground.

The signal travels down the wires to each component in the circuit, passing through each one along the way. The I2C protocol is unique because all components are connected to the same data and clock lines. To manage communication on a shared line, the signal output from the microcontroller contains two parts: an address (or ID) and the command itself. Each component has a unique address, and when a signal with a matching address passes by, the component

listens to the following command and executes the instruction. This combination of address and command is what enables I2C to function efficiently, making it a more effective cabling method for multiple devices.

While there are other bus protocols that can effectively communicate data, we have chosen to use I²C for this application. For example, SPI employs four wires and offers faster communication but is more cumbersome, complex, and supports only a limited number of devices. Additionally, asynchronous serial communication could be utilized; however, it is designed for point-to-point communication with a single device, making it less suitable for our specific needs. As stated in a SparkFun article, "I²C requires a mere two wires, like asynchronous serial, but those two wires can support up to 1008 peripheral devices. Also, unlike SPI, I²C can support a multi-controller system, allowing more than one controller to communicate with all peripheral devices on the bus" [15].

The communication protocol leverages I2C buses to streamline system implementation. These buses provide two connection paths that link all elements along a given line, allowing signals to be sent with a unique ID and command for each element. This approach is highly beneficial as it minimizes the wiring required between components, optimizing the circuit's layout. Additionally, it enhances circuit efficiency by reducing signal paths and enabling faster, more straightforward communication between elements.

4.4.2. PCB Board

To ensure reliable connections, a PCB will be used to maintain secure links from the microcontroller to the wire outputs. Components will be either directly soldered to the board or connected via adapter pins, facilitating easy replacement in the event of a component failure. Additionally, cut-to-length wires will be used to prevent tangling and ensure stable attachment, a critical factor in maintaining a clean, organized layout. This arrangement is essential for reliable operation, as it minimizes the risk of disconnections or damage when the ToppleBot rotates and shifts between orientations.

5. Telecommunication

Keeping with the theme of improving upon past designs, like Cubli, REM-RC, and CUBEBAS, ToppleBot will feature a display of the robot's current state, as well as wireless control capabilities. Of the three examples mentioned, CUBEBAS [1] made the most progress in this endeavor, using a COM port to display the cube through MATLAB. Their configuration, while showing initial success, did fail after "some time running" and used transformation matrices that often showed an unusual offset. ToppleBot will take a similar, but more supported and common approach to real-time robot visualization.

To fulfill the requirements of the real-time wireless display and control, a telecommunication system is needed. The following sections will delineate the choices made to fulfill the requirements of the telecommunication system. More specifically, Section 5.1 determines the overall telecommunication framework, while Sections 5.2 and 5.3 describe the software stack and microcontroller selection, respectively.

The choices made in the following section are made entirely with the system requirements as the primary goal. The following table outlines the specific system requirements, measures of performance and justifications for the telecommunication system for the ToppleBot project.

Table 7: Telecommunication System Requirements									
Number	**Requirement**	**Measure of Performance**		**Justification**	**Owner**	**Validation**	**Priority**	**Status**	**Remarks**
		Threshold	**Objective**						
7.1	All communications between the control station and the cube rover must be performed wirelessly.	Wireless Communication	Wireless Control	Wired systems would inhibit the functionality of the cube rover. Wireless programming would allow for quicker prototyping.	Gabriel	Physical Testing	1	In Progress	
7.2	The chosen transmission frequency should bypass line-of-sight obstructions and provide adequate range.	10+ meters	50+ meters	Allows for a stationary control station to be set up away from the cube rover.	Gabriel	Physical Testing and Technical Specifications	2	Not Started	
7.3	The telecommunications system must maintain consistent performance regardless of orientation.	TBD	TBD	The cube's orientation will change, so it's crucial that communications remain unaffected.	Gabriel	Physical Testing and Technical Specifications	1	Not Started	

5.1. Telecommunication System Options

There are various options for communication between a robot and a control station. The following table introduces some of the most common approaches, as well as their advantages and disadvantages for the ToppleBot's operating conditions.

Table 8: Telecommunication System Framework Options

Option	Advantages	Disadvantages	Conclusion
Wi-Fi	- High-bandwidth - Widely compatible with robotic software - Adequate range for indoor applications	- Relatively high power consumption - Can be susceptible to interference based on orientation - Often limited to indoor applications	Good option for moderate range, complex communications
Bluetooth	- Lower power consumption - Can use COM port (like CUBEBAS) - Good for short range communication	- Limited range - Low-bandwidth (compared to Wi-Fi) - Resistant to interference based on orientation	Good option for short range, simple communications
LoRa	- Long range capabilities - Low power consumption - Resistant to disturbances	- Low-bandwidth - Low compatibility with robotic software	Good option for long range, simple communications
Zigbee	- Lower power consumption - Medium range capabilities - Reliable on crowded networks	- Moderate-bandwidth - Low compatibility with robotic software	Good option for medium range, multi-agent systems

Considering the available options, Wi-Fi is evidently the most applicable. With its high-bandwidth communication, the burden of data optimization (a challenge faced by CUBEBAS) will be lifted, allowing the ToppleBot's communications to not be limited by the telecommunication data transfer. Additionally, especially during the first 8 months of the project, most of the testing will take place indoors, and with great connectivity to Wi-Fi. Thus, the team will be able to most easily and effectively communicate with the cube over Wi-Fi. Lastly, common robotic software has support for Wi-Fi as many robotic applications are indoors.

As mentioned in Table 7, Wi-Fi does have some disadvantages. Of these, the most concerning is the possibility of communication interference due to the robot's orientation, surrounding structures, and instrumentation. To meet requirement 6.3, it is important that these shortcomings do not affect the system's performance. Thus, if significant delays or complications arise, though unlikely at short ranges, the altering of antenna geometry to minimize obstructions will be explored.

5.2. Communication Software Stack

Based on the choice to use Wi-Fi, a more informed decision could be made about the necessary software stack to support the chosen telecommunication protocol. Additionally, as mentioned in the structural and electrical discussion, the onboard hardware is limited to a microcontroller, with lower RAM and less robotic software compatibility than larger and more expensive single-board computers. Thus, the software used must be compatible with microcontrollers and support Wi-Fi communication. It must also support complex data types for real-time display and control.

With the aforementioned considerations and additional tools for robotics, Micro Robot Operating System (Micro-ROS) was chosen. To accurately introduce Micro-ROS concepts and the reasons behind the decision to use Micro-ROS, ROS must first be introduced. The Robot Operating System currently has 2 iterations, ROS and ROS2. It is built off of DDS (multiple implementations) as the middleware layer and UDP as the transport layer [3]. These sub-systems allow for a publish/subscribe communication framework to allow for different nodes (normally Python or C++ executables) to communicate with each other. Micro-ROS allows access to this same system onboard the limited memory microcontrollers. Micro-ROS offers seven key features, as defined by the developers [4]:

1. Microcontroller-optimized client API supporting all major ROS concepts
2. Seamless integration with Robot Operating System 2 (ROS2)
3. Extremely resource-constrained but flexible middleware
4. Multi-RTOS support with generic build system
5. Permissive license
6. Vibrant community and ecosystem
7. Long-term maintainability and interoperability

In the Micro-ROS framework, the microcontroller acts as a node, with the capability to publish and subscribe to different topics. In this configuration, Wi-Fi acts as the network layer, enabling other systems to communicate with the microcontroller wirelessly. The ToppleBot will make use of this wireless communication. Additionally, as detailed in feature 2, Micro-ROS integrates seamlessly with ROS2, allowing the control station to use ROS2 to communicate with the cube. Lastly, if we do need to switch between different microcontrollers, Micro-ROS supports panoply of microcontrollers.

5.3. Microcontroller Options

Based on the choice to use Micro-ROS, a more informed decision could be made about the necessary microcontroller to support the ToppleBot's telecommunication needs. Luckily, Micro-ROS lists the supported microcontroller on their website. To maximize scalability the hardware selection also considers the size of the microcontroller, aiming to have the smallest packaging for the smallest possible cubes. The following table contains some of the possible options that met the size requirements and Micro-ROS compatibility:

Table 9: Microcontroller Options			
Option	**Key Features**	**Supported Transports**	**Conclusion**
Espressif ESP-32	- MCU: ultra-low power dual-core Xtensa L13 - RAM: 520 kB - Flash: 4 MB	- UART - Wi-Fi UDP - Ethernet UDP	Suitable for wireless connection.
Arduino Potenta H7	- MCU: Dual-core Arm Cortex-M7 and Cortex-M4 - RAM: 8 MB - Flash: 16 MB	- UART - Wi-Fi UDP	Suitable for wireless connection.
Raspberry Pi Pico RP2040	- MCU:Dual-core Arm Cortex-M0+ - RAM: 264 kB - Flash: up to 16 MB	- UART - USB	Not suitable for wireless connection.
Teensy 3.2	- MCU: ARM Cortex-M4 MK20D956VLH7 - RAM: 64kB - Flash: 256 kB	- UART - USB	Not suitable for wireless connection.
Teensy 4.0/4.1	- MCU: ARM Cortex-M7 iMXRT1062 - RAM: 1024 kB - Flash: 2048 kB	- UART - USB - Ethernet UDP (4.1)	Not suitable for wireless connection.

It is imperative that the microcontroller aboard the ToppleBot can directly interface with Wi-Fi, thus, from Table 8, it is evident that the only two options for the ToppleBot microprocessor are the Espressif ESP-32 and the Arduino Potenta H7. Due to the minimal expected load on the onboard microprocessor, the RAM is largely ignored. Additionally, CUBEBAS and REM-RC both used the Espressif ESP-32, giving an example for how the microcontroller may be set up for this application. Thus, the Espressif ESP-32 was chosen as the microprocessor for the ToppleBot. As it is supported by Micro-ROS, it will easily integrate into the chosen telecommunication system.

With the complete telecommunication system built off of Micro-ROS and the Espressif ESP-32, the cube will be able to broadcast messages by publishing to ROS2 topics, and receive commands by subscribing to ROS2 topics. Thus, the system will meet the requirements detailed in Table 9.

6. Control Station

As mentioned briefly in the overall system requirements, the control station must be able to display the Topplebot's current state and send behavioral command messages to the ToppleBot. Additionally, in tandem with the sizing algorithm that will allow others to build their own ToppleBot iterations, this control station must be easily deployed on various workstations with minimal set-up and start-up time. The specific requirements, measures of performance and justifications for the control station are detailed in Table 9.

The following subsections detail the methods for meeting the requirements defined in Table 9. More precisely, Section 6.1 describes the overall system architecture, accounting for constraints and tools set by the telecommunication system. Section 6.2 covers the visualization system and software tailored to the ToppleBot, while Section 6.3 presents the behavioral command creation that can be visually verified on the display. Lastly, Section 6.4 discusses how the ToppleBot documentation will maximize workstation compatibility and minimize start-up and setup times for future users.

Table 10: Control Station System Requirements									
Number	**Requirement**	**Measure of Performance**		**Justification**	**Owner**	**Validation**	**Priority**	**Status**	**Remarks**
		Threshold	**Objective**						
10.1	The control station must receive sensor data from the cube rover throughout the entire duration of its operation.	N/A	N/A	Necessary for system monitoring and troubleshooting.	Gabriel	Time-Stamped Transmission Logs	1	In Progress	Currently receiving random integer messages.
10.2	The control station must utilize any received sensor data to display the cube rover's state and the condition of all relevant subsystems.	Command Line Interface	Graphical User Interface	Necessary for user interface.	Gabriel	Observation Agreement of Cube State and Displayed State	1	In Progress	Onboard microprocessor is receiving IMU readings.
10.3	The control station must be able to transmit commands to modify the cube rover's behavior.	Control Algorithm Selection	Desired Control Angle Selection	Necessary for user control over the cube rover.	Gabriel	Observation of Cube Behaviour	1	Not Started	
10.4	The control station should be deployable on a standard workstation with minimal modifications.	N/A	N/A	Reduces reliance on specific hardware, allowing for easier scalability.	Gabriel	Deployment on Multiple Stations	2	Not Started	
10.5	The control station deployment should be documented such that future users can easily deploy it.	1 Hour Start-Up	20 Minute Start-Up	Allows for easier replication by similar project groups.	Gabriel	Timing of Deployment on Multiple Stations	2	In Progress	Directions will be housed in GitHub's ReadMe.

6.1. Control Station Configuration

To meet the system requirements, the control station must be able to interface with the telecommunication system, detailed in Section 5 of this report. By implementing Micro-ROS for telecommunication, the choice of control station software was effectively determined, since Micro-ROS can only interface effectively with ROS2. Thus, the control station will use a ROS2 installation with a Micro-ROS agent to bridge the gap between the micro-ROS communication and the onboard ROS2 installation. The decision to use Micro-ROS factored for the integration of ROS2, and allows the ToppleBot to make use of ROS2's plethora of tools for robotics.

The control station must be able to display the current state of the ToppleBot, while also sending high-level behavioral commands to it. Both of these requirements will, in part, be facilitated by the use of ROS2 topics. Through the telecommunication system, the cube and control station will be able to subscribe to topics published by the respective systems. More specifically, the cube will publish information about its orientation such that the control station can display the cube, while the control station will publish behavioral commands such that the cube can act on control station user input. The following sections will detail the specific methods for both the visualization and command generation as they apply to the control station.

6.2. Visualization Configuration

Before detailing how the control station will visualize the ToppleBot state, the need for a control station must be discussed. With the increasing computational capabilities of microcontrollers and single-board computers, it is important to justify running computational efforts away from the system and increasing its complexity. This justification lies in ToppleBot's efforts in scalability and budget considerations. To maximize its scalability, the ToppleBot must minimize hardware component size and weight. Thus, a single-board computer, like the Raspberry Pi, that could possibly run all necessary computation for visualization, would prove too large and expensive for this application. Instead, the ToppleBot is using the Esperessif ESP-32 with just 520 [*kB*] of memory. With such limited memory, visualizations must be exported to a more capable system, which will be the control station.

Thus, the onboard system will export its orientation information in a ROS2 topic named “orientation” with an IMU message format. The IMU message, as defined by ROS2 developers, is formatted as seen in Table 10 [5]. Note that the current IMU configuration returns Euler angles, and the onboard system will need to convert the angles to a quaternion.

Table 11: IMU Message Format	
Message Component Format and Data Type	**Message Component Name**
std_msgs/msg/Header header - uint32 - time - string	header - seq - stamp - frame_id
geometry_msgs/msg/Quaternion - float64 - float64 - float64 - float64	orientation - x - y - z - w
float64[9]	orientation_covariance
geometry_msgs/msg/Vector3 - float64 - float64 - float64	angular_velocity - x - y - z
float64[9]	angular_velocity_covariance
geometry_msgs/msg/Vector3 - float64 - float64 - float64	linear_acceleration - x - y - z
float64[9]	linear_acceleration_convariance

While most of the entries are self explanatory, and are common for a 9 DOF IMU, the covariances and header are unique to the ROS2 implementation. As can be expected, the covariances are a measure of the uncertainty of the measurements. It is unlikely that these will be used for a singular IMU configuration, because they are normally used for the odometry system to know which IMU's to weigh more or less. In more developed iterations, especially those at larger scales, the incorporating of various IMU's can be explored for maximum accuracy. The header, on the other hand, contains the frame identification and other ROS2 specifics which are used in visualization to ensure the correct object is displaying the transformations.

Subscribing to the "orientation" topic with IMU message type carrying the IMU information, the control station will develop transfer frames using ROS2's tf2 system [6]. In this system, there exists a world frame and a robot frame. The world frame will be taken as the robot's initial orientation and position, while the robot frame will be constantly updated using the IMU information and a node creating the transforms. The transform messages contain the following information:

Table 12: Transform Frame Message Format	
Message Component Format	**Message Component Name**
std_msgs/msg/Header - uint32 - time - string	header - seq - stamp - frame_id
string	child_frame_id
geometry_msgs/msg/Transform - geometry_msgs/Vector3 - float64 - float64 - float64 - geometry_msgs/Quaternion - float64 - float64 - float64 - float64	transform - translation - x - y - z - rotation - x - y - z - w

The transforms are assigned to the child frame, which will be the robot's frame. For initial testing, the translation will not be accounted for, only accounting for the rotation derived from the quaternion. To meet the odometry required (translation tracking), a mathematical model will be made of the system (delineated in Section 7) from which odometry can be derived.

Importantly, to accurately display the cube in RViz, a model of it will be integrated. There are two components of the modeling framework for RViz. There is a URDF (Unified Robot Description Format) file written in XML that carries the physics and frame ID's of the robot, while STL files with the meshes of the robot are used for visualization. Thus, with the same CAD models used for manufacturing, the robot can be displayed in RViz with accurate physics described by the URDF file. For reference, an RViz example display [7] is shown below. In the future, the robot will be modeled and displayed in this window on the control station, meeting the display requirements:

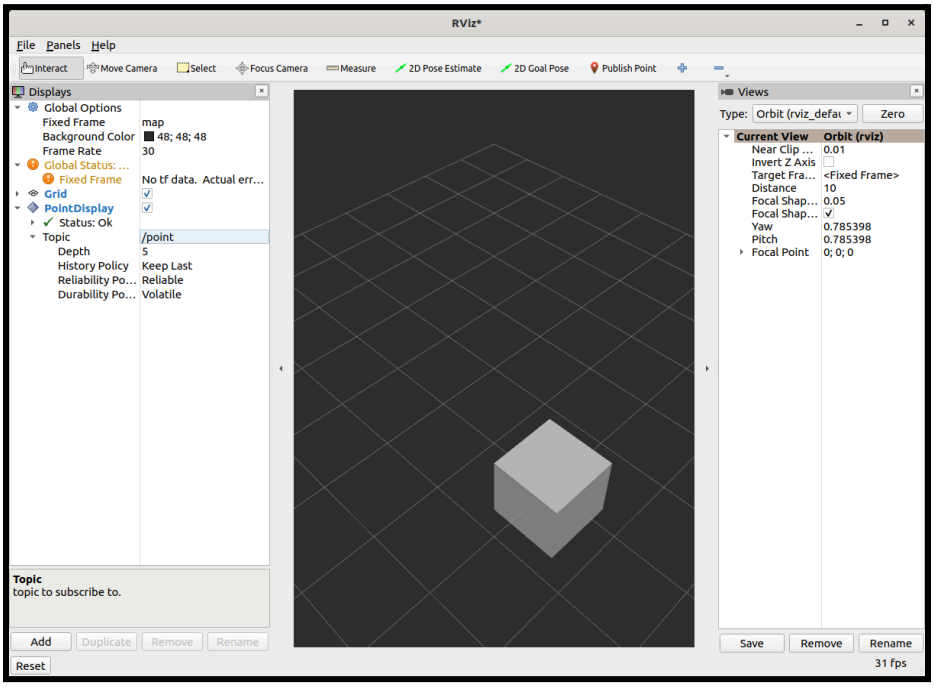

Figure 7: Example RViz display with initial cube for reference

6.3. Command Creation and Communication

To send commands to the ToppleBot from the control station, ROS2 actions will be utilized. Using nodes in the ROS2 installation of the control station, actions can be called, providing continuous feedback on the current state of the action until success or failure. Thus, an action will be created for each of the behaviors: edge_balance, corner_balance, kip-up, and topple. The definition and calling of the actions will be exclusively defined within the control station, removing computational load from the onboard system.

The actions will define desired angular velocity and orientations for the control algorithm to target, and the IMU will send feedback on action completion of failure. The control goals, defined as ROS2 Vector3 and Quaternion messages to the "control_goal_ang_vel" and "control_goal_quaternion" topics will be communicated to the cube which will convert such control goals to motor commands. Initially, calling actions will be through the command line, but further in development, a GUI will be developed using Python for more intuitive action calling.

6.4. Documentation and Compatibility

The final important consideration for the control station is minimizing start-up and set-up times while maximizing work station compatibility. On Linux systems the start-up and set-up optimization process is trivial. With a guide incorporated into the GitHub repository, a bash script, written by the ToppleBot team, can be called to ensure all the necessary command line arguments are passed within seconds. However, significant complexity arises when accounting Windows workstations. To support Windows, additional documentation with directions for using WSL (Windows Subsystem for Linux) will be provided such that the future users of ToppleBot can deploy the system on their respective workstations within reasonable time limits.

7. Control System

Accurate orientation determination is essential for the closed-loop feedback control of the cube rover. This project employs a 9-DOF IMU that integrates an accelerometer, gyroscope, and magnetometer to effectively track the rover's orientation. Section 7.1 addresses the challenge of sensor drift, detailing methods employed to mitigate this issue through the use of reference points and filtering techniques. Section 7.2 reviews the technical specifications of the selected sensors, comparing them to those utilized in ETH Zurich's Cubli and justifying the choice of a 9-DOF IMU by addressing potential concerns. Finally, Section 7.3 evaluates the necessity of multiple IMUs to achieve robust performance, aiming to enhance the modeling framework established in prior research while implementing mechanisms for autonomous self-balancing. Collectively, these components lay a solid foundation for a cube rover capable of autonomously maintaining its balance in dynamic environments, with specific performance measures outlined in Table 6.

Table 13: Control System Requirements								
Number	**Requirement**	**Measure of Performance**		**Justification**	**Owner**	**Validation**	**Priority**	**Status**
		Threshold	**Objective**					
13.1	The cube rover must be equipped with an IMU with adequate sample rate and precision.	TBD	TBD	Accurate and current data is essential for effective control calculations.	Kilian	Technical Specifications	1	In Progress
13.2	The cube rover controller must transmit control signals with minimal latency.	TBD	TBD	Delays in actuation reduce that functionality of the cube rover.	Kilian	Technical Specifications	1	In Progress
13.3	Each reaction wheel must generate enough torque to rotate the cube around its edge.	Single Rotation	Multiple Rotations	Necessary to perform any locomotion or "kip-up" behaviors.	Kilian	System Modelling and Physical Testing	1	In Progress
13.4	The cube rover should be able to balance on any edge.	Starting on Desired Edge	Starting on Tangential Face	Mission Objective	Kilian	Physical Testing	1	Not Started
13.5	The cube rover should be able to balance on any corner.	Starting on Desired Corner	Starting on Tangential Edge	Mission Objective	Kilian	Physical Testing	1	Not Started
13.6	The cube rover should be able to reject external disturbances and maintain stability.	TBD	TBD	Mission Objective	Kilian	Physical Testing	1	Not Started

7.1. Orientation Determination

Accurately determining the state of the cube rover requires the integration of appropriate sensors. A 9-DOF IMU is selected to meet this requirement, providing full orientation tracking. A 3-axis accelerometer measures the accelerations experienced by the cube, allowing partial orientation estimation relative to the Earth's surface by identifying the direction of gravitational acceleration. However, this alone is insufficient to fully define the cube's orientation, as it provides information about precession and nutation but leaves the spin around an axis normal to the Earth's surface unresolved.

A 3-axis gyroscope solves this problem by measuring angular velocity, which can then be integrated to determine the cube's spin, giving a complete orientation profile. Together, the accelerometer and gyroscope account for 6-DOF. While this setup seems sufficient for orientation determination, it falls short in real-world applications due to sensor drift. Over time, inaccuracies in the gyroscope data accumulate, causing the sensed orientation to deviate from the true state. This drift, which often gets overlooked when moving from simulation to physical implementation, becomes a significant issue in long-term operation.

Although increasing the sampling rate could reduce drift to some extent, it only goes so far. A more reliable solution is to introduce a stable reference point. Military inertial navigation systems manage drift by validating location data with GPS. When GPS is unavailable, the system begins to drift. In our case, however, we seek to mitigate the orientation drift of the cube and not the position drift. For this we introduce the final 3-DOF through a 3-axis magnetometer. By measuring the Earth's magnetic field, the magnetometer offers a stable reference to determine the cube's true orientation relative to the magnetic poles. With data from all three sensors and the use of appropriate filtering algorithms, sensor drift can safely be ignored.

7.2. IMU Selection

The technical specifications of these sensors are equally important. ETH Zurich's Cubli used an ADXL345 accelerometer from Analog Devices and IDG-500 and ISZ-500 gyroscopes from InvenSense [8]. The ADXL345 is a 3-axis digital accelerometer with a full-scale range of ±2g and a maximum sampling rate of 3200 Hz [9]. The IDG-500 is a 2-axis digital gyroscope with a full-scale range of 110°/s and a sensitivity of 9.1 mV/°/s [10]. The ISZ-500 is a single-axis gyroscope with equivalent capabilities [11]. It is worth noting that Cubli did not implement a magnetometer in their design. One possible reason for this could be attributed to short operating times. A more likely reason, however, lies in the close proximity of the motors with the IMU. It is possible that these motors interfered with the magnetometer and drowned out the sensing of the Earth's magnetic field. This theory would need to be confirmed with further testing.

Despite these challenges, the decision to use a 9-DOF IMU over a 6-DOF unit is justified, as the cost difference is negligible and both sensor packages share the same form factor. Additionally, the increase in data transmission size is minimal, with only 2 bytes required to transmit magnetometer data. For our design, we prioritized selecting IMU units that were readily available on campus. Initially, we selected the Bosch BNO055, as its performance is comparable to the ADXL345 and the InvenSense gyroscopes used in Cubli. However, during testing, we encountered a known issue where the BNO055 violates the I2C communication protocol when interfacing with the ESP32 microcontroller. This incompatibility would require us to switch to a

different microcontroller to resolve the issue. Instead, we transitioned to the InvenSense MPU9250, which provides similar functionality and is fully compatible with the ESP32, allowing us to move forward with the design without major changes to the system architecture.

7.3. System Modelling and Control

The Cubli utilizes six IMUs to accurately determine its orientation [8, 12]. In contrast, the design implemented by REM-RC, which CUBEBAS replicates, employs only one IMU for achieving self-balancing behavior [13]. This approach, however, lacks practicality, as the desired equilibrium orientations are hardcoded into the microcontroller's firmware. Consequently, CUBEBAS can only maintain balance at predefined orientations rather than dynamically adjusting to balance on any edge or corner.

This limitation raises an important question: how many IMUs are truly necessary to create a system that can demonstrate similar capabilities to the Cubli? To explore this, the thesis "Modeling and Non-Linear Attitude Control Utilizing Quaternions" by Fabio Bobrow provides relevant insights [14]. Bobrow's work demonstrates state estimation comparable to the Cubli, relying on a single IMU for system modeling. However, it is crucial to note that this design does not incorporate a braking mechanism, which is essential for enabling autonomous self-balancing.

Thus the final goal of this project is to validate and extend Bobrow's system model and control strategies while integrating a mechanism that allows for autonomous self-balancing behavior. This effort seeks to refine the design of the Cubli by removing unnecessary complexities and confirming the robustness of Bobrow's modeling approach. By doing so, a more streamlined and efficient system can be developed, enhancing performance while reducing necessary hardware.

8. Conclusion

ToppleBot aims to create a scalable and adaptable robotic platform for autonomous self-balancing behaviors, with significant progress made toward establishing its system architecture. The selection of materials, motor control systems, and the development of an effective braking mechanism are currently being explored to ensure the rover meets its operational requirements.

Future steps will focus on refining these designs, validating the control algorithms through simulation and physical testing, and ensuring the components can withstand real-world conditions. The project’s success will hinge on achieving stable, autonomous balancing while maintaining scalability, allowing for further adaptation and testing across different environments and use cases. As development continues, frequent testing will be crucial to validate the system's performance and the cube rover's capabilities.

9. References

[1] D. Harrington, “Project CUBEBAS: Cube Balancing and Stabilization Final Design Report,” Embry-Riddle Aeronautical University, May 2, 2023. Accessed: Nov. 02, 2024.

[2] M. Gajamohan, M. Merz, I. Thommen, and R. D'Andrea, "The Cubli: A cube that can jump up and balance," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, Oct. 2012, pp. 3722–3727. doi: 10.1109/IROS.2012.6385896.

[3] ROS, “ROS 2 Documentation: Humble,” Accessed: Nov. 02, 2024. [Online]. Available: https://docs.ros.org/en/humble/index.html

[4] Micro-ROS, “Overview and Supported Hardware,” Accessed: Nov. 02, 2024. [Online]. Available: https://micro.ros.org/docs/overview/hardware/

[5] ROS, “sensor_msgs/Imu Message Documentation for ROS2 Foxy,” Accessed: Nov. 03, 2024. [Online]. Available: https://docs.ros2.org/foxy/api/sensor_msgs/msg/Imu.html

[6] ROS, “Introduction to tf2,” Accessed: Nov. 03, 2024. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Intermediate/Tf2/Introduction-To-Tf2.html

[7] ROS, “Building a Custom RViz Display,” Accessed: Nov. 03, 2024. [Online]. Available: https://docs.ros.org/en/humble/Tutorials/Intermediate/RViz/RViz-Custom-Display/RViz-Custom-Display.html

[8] M. Gajamohan, M. Merz, I. Thommen, and R. D’Andrea, “The Cubli: A cube that can jump up and balance,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal: IEEE, Oct. 2012, p. 3722–3727. doi:10.1109/IROS.2012.6385896.

[9] Analog Devices, “3-Axis, ±2 g/±4 g/±8 g/±16 g Digital Accelerometer,” ADXL345 Datasheet, 2009.

[10] InvenSense, “Integrated Dual-Axis Gyro,” IDG-500 Datasheet, 2008.

[11] InvenSense, “ISZ-500 Single-Axis Z-Gyro Product Specification,” ISZ-500 Datasheet, 2008.

[12] S. Trimpe and R. D’Andrea, “Accelerometer-based tilt estimation of a rigid body with only rotational degrees of freedom,” in 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK: IEEE, May 2010, pp. 2630–2636. Doi: 10.1109/ROBOT.2010.5509756.

[13] Remigijus, remrc/Self-Balancing-Cube. (Sep. 08, 2024). C++. Accessed: Sep. 09, 2024. [Online]. Available: https://github.com/remrc/Self-Balancing-Cube

[14] “The Cubli: Modeling and Nonlinear Attitude Control Utilizing Quaternions | IEEE Journals & Magazine | IEEE Xplore.” Accessed: Nov. 03, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/9524577.

[15] “I2C,” I2C - SparkFun Learn. Accessed: Nov. 4, 2024. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c/all#why-use-i2c