RailRider: Uni-Wheel Self-Balancing Robot

By

James Recera

Varun Sharma

Zhanshuo Zhang

Design Document for ECE 445, Senior Design, Spring 2026

TA: Abdullah Alawad

19 February 2026

Project No. 75

Contents

1. Introduction

1.1 Problem

A lot of important industrial inspection locations are basically “thin-structure environments” where a normal robot is awkward or unsafe: narrow beams, cable trays, pipe-rack edges, and long tunnel-like spaces. These places show up in real industrial environments like data centers cable management, HVAC/ventilation runs in industrial facilities, and even some space missions like lunar or martian tunnel scout, where falling off an edge could mean mission failure. A typical RC car is too wide and needs a turning radius, and a drone can be loud, have a short battery, and often not be allowed indoors nor functioning well in a confined area. We want a compact platform that can stably move on narrow structures and produce some useful inspection results.

1.2 Solution

We will build a reaction-wheel stabilized uni-wheel robot that can travel along a narrow beam/rail while carrying a camera-based perception payload. The core idea is that the robot can balance itself with a tiny contact footprint, so it can ride on structures that would make a 4-wheel robot fall off. Overall, the robot will consist of a drive wheel and a flywheel that rotate orthogonal to each other, utilizing conservation of angular momentum to balance itself against gravity. Each of the wheels will be driven by brushed or brushless motors along with motor drivers, and some sensors and encoders monitor and record the rotation speed of the wheels constantly. In order for the robot to stabilize itself robustly, we need to provide enough power with a central battery management system that will deliver power to other subsystems safely, and then implement several robust feedback control loops algorithms on the microcontrollers that receive input information from sensors and output intended voltage to motor drivers.

After achieving the goal for the robot to balance itself on tiny contact, we will implement several computer vision algorithms on the camera-based perception subsystem that communicate with the center microcontroller and output videos to phones or laptops. From these videos, users of this robot could make decisions and remotely control the forward or backward motion of the robot. The camera system with the CV algorithms will have two main functions. Firstly, while the user will drive the robot forward or backward on a narrow path using the software interface we develop on the computer, the camera will detect edges or obstacles on the path and provide a safe override to prevent falloff, so the robot can be driven on any narrow path without complicated human maneuvering. Another function of the camera system is to perform assisted inspections by utilizing object identification and classification from the computer vision. The robot follows path directions and uses perception cues and logs simple “inspection events” for different types of missions. For example, these events can be broken cables for data centers, debris or blockages for ventilation in industrial facilities, etc.

1.3 Visual Aid

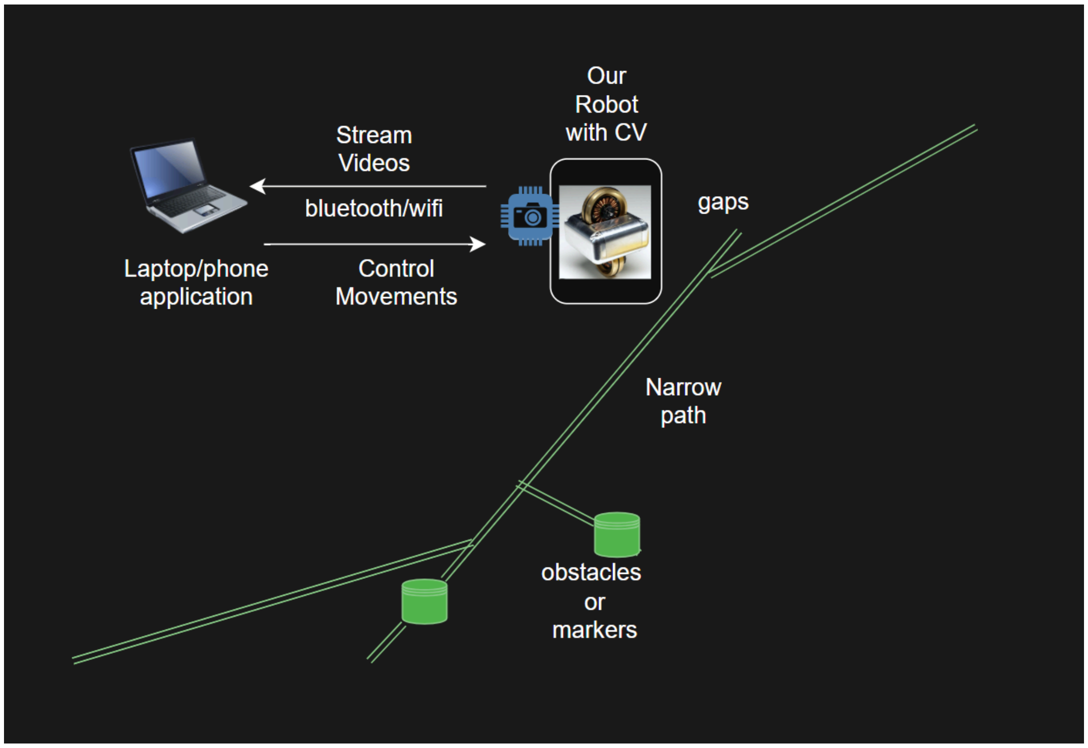

Figure 1. Visual Aids for our Wheelbot project and demonstration

1.4 High-Level Requirements

We pick the three out of four following quantitative characteristics to show success within our project (one of the objectives- 3rd or 4th- might be optional depend on the difficulty of the task):

1. Balance: The robot shall demonstrate stable and autonomous self-balancing on a stationary surface for at least 60 seconds without any external support or manual intervention, maintaining an upright attitude throughout the trial.
2. Straight narrow-structure driving: The robot shall be able to drive on a 1 m long straight rail/beam path. The robot, under the user's control and under self-balancing and stabilizing, should maintain continuous contact with the path without falling off.
3. Twisted narrow-structure driving with branches, gaps, and obstacles that represent the intended operating environment. Under this task, the system shall implement a perception-driven safety override that detects hazards like obstacles and gaps along the

path and makes an emergency stop so that the robot comes to rest before the hazard, with a maximum stopping distance of 20 cm.

4. Inspection output: During each drive, the robot shall stream live video to a laptop or phone and generate a structured log containing at least three vision-detected inspection events, such as “marker reached or tag detected,” “edge or drop-off detected and stop triggered,” and “obstacle detected and stop triggered.” If time permits, an additional event type may be included, such as “debris or blockage flagged” or “loose or broken cable flagged.”

2. Design

2.1 Block Diagram

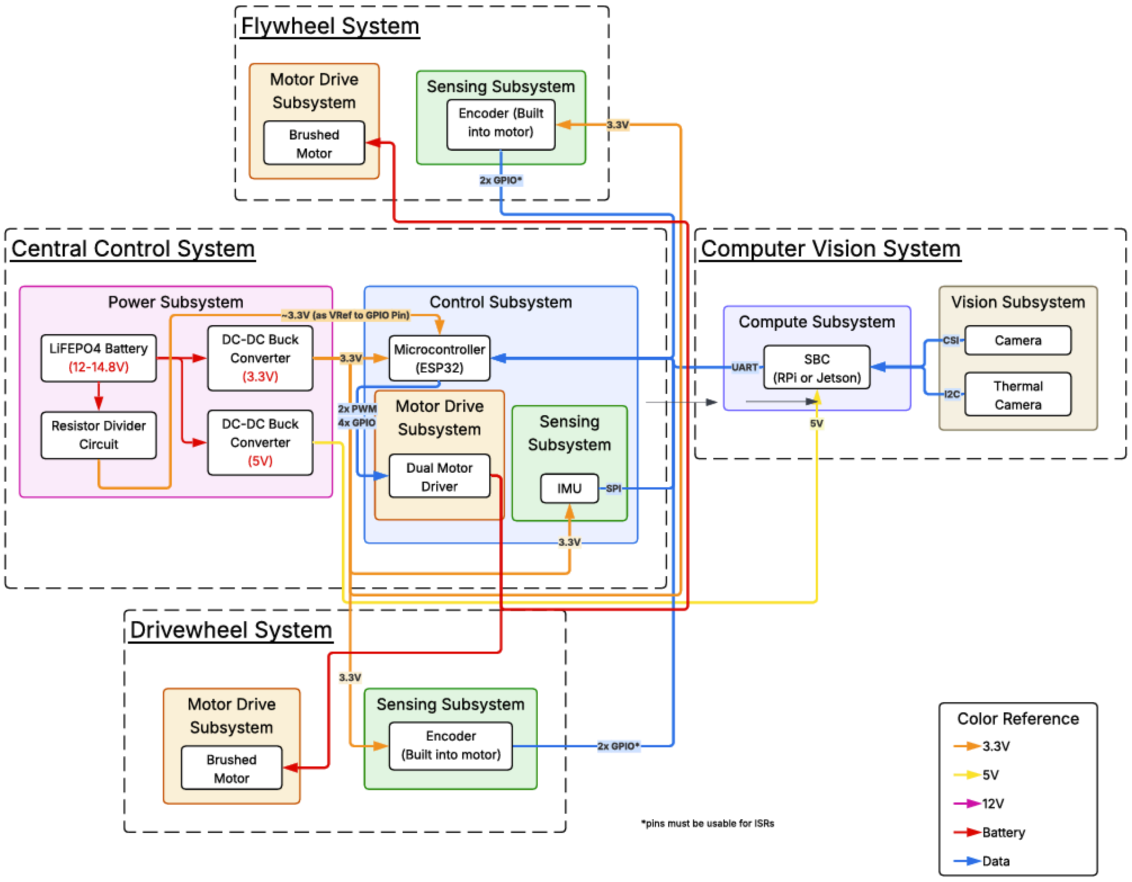

Figure 2. High-level block diagram of each system within RailRider

2.2 Subsystem Overview

2.2.1 Power Subsystem

Each individual subsystem within our robot requires different amounts of power, which necessitates the need for a specialized subsystem that can provide each of our required voltages. The robot must be fully mobile in order to accomplish meaningful tasks such as traversing down a beam and exploring different areas, thus introducing the need for a battery and a safe way of managing it. Without the power subsystem, components would not be able to receive any power, and thus it is a key component to the functioning of the rest of the robot.

The main constraints here were weight, cost, current output, and safety. We considered NiMH first, but it was too heavy and did not provide enough current for our motor demands. We also considered LiPo, but decided against it because of safety concerns, added charger cost, and limited team experience using LiPo batteries. We selected LiFePO4 as a middle-ground option because it gives better current capability than NiMH while being safer to use and charge than a typical LiPo setup. We use XT60 connectors for the battery connection, fuses for protection, capacitors for smoothing, and off-the-shelf buck converters for 5V and 3.3V because they are practical for our use case and include built-in protections.

the LiFePO4 battery (12–14.8V) is the main source. Battery voltage is routed directly to the motor drive path and the motors. The dual motor driver we are using accepts the external motor power separately (P+ / P-) and uses a 3.3V–5V compatible signal control interface, which matches our plan to power the control side from 3.3V while passing battery voltage to the motor power path. In addition, the buck converters generate 5V and 3.3V for electronics. The compute subsystem SBC (RPi or Jetson) uses 5V, and the ESP32-based control subsystem, IMU, encoder sensing interfaces, and the dual motor driver control logic use 3.3V. We also have a resistor divider circuit in the power subsystem whose output routes to the esp32, which is used to monitor battery voltage, so we know when the battery needs to be charged.

Table 1: Power Subsystem – Requirements & Verification

Requirements	Verification
• The power subsystem must provide 12V battery power for the motor drive path, 5V for the SBC (RPi or Jetson), and 3.3V for the ESP32, IMU, encoder interfaces, and motor driver control logic. • The 5V and 3.3V outputs must stay within +/- 0.2V of nominal during normal robot operation.	• Ensure the robot is powered on in an idle state. Measure the battery output, 5V output, and 3.3V output at the subsystem outputs. Record the values. Confirm the 5V output is within 4.8V–5.2V and the 3.3V output is within 3.1V–3.5V. • Then, run the robot in active conditions (SBC on, ESP32 on, sensors active, motors enabled). Measure again and confirm the outputs remain in spec.
• The power subsystem must not cause brownouts or unexpected resets while the SBC, ESP32, sensors, and motor drive system are active at the same time.	• Ensure the robot is in a normal operating condition with the SBC, ESP32, and sensors active. Then command motor motion. Observe the system during the test. Confirm there is no SBC reboot, no ESP32 reset, and no sensor dropout. If a reset or dropout occurs, record the test condition and mark the

	requirement as failed.
• The power subsystem must include a fused battery path and a keyed battery connector (XT60 or equivalent) for safe battery connection during normal operation and repeated testing.	• Inspect the battery in the system power path. Confirm that a fuse is installed in series with the battery supply path.
• The power subsystem must monitor battery voltage so that the system can determine when the battery should be recharged.	• Power on the system and read the battery monitor value from the resistor divider measurement path using the intended controller readout or telemetry output. Record the reported value. • Measure the battery voltage directly with a meter and record the value. Confirm the monitored value matches the direct measurement closely enough to make a low battery decision. • Change the battery voltage condition if possible and confirm the monitored value changes in the same direction.
• The buck converters must supply the required electronics power without shutting down during continuous operation.	• Run the robot with the SBC, ESP32, and sensors active for 10 minutes. At the end of the test, measure the 5V output and the 3.3V output. Confirm both outputs are still within the required ranges. Confirm that no buck converter shut down during the test.

2.2.2 Motor Drive Subsystem

The Motor Drive Subsystem, which includes both the drive wheel motor path and the flywheel motor path, is responsible for rotating the wheels that let the robot move and maintain balance. Each motor is driven by a motor controller that receives commands from the MCU. These commands are based on data from the sensing subsystem, including the encoders and IMU, which are then used by the control algorithm to determine motor speed and direction. This subsystem has two separate motor roles. The drive wheel motor must provide enough torque to move the robot. The flywheel motor must provide high speed rotation for balance control and help recover the robot back upright. Because these two jobs are different, the motor selections and mechanical interfaces are also different.

Our main design constraints here were torque, speed, current draw, cost, weight, and mechanical compatibility. For the drive wheel, we selected a geared brushed motor with an encoder because it provides good torque, includes speed feedback, and matches our mechanical interface well. For the flywheel, we selected a high-speed motor that directly matches the chosen goBILDA shaft and hub ecosystem, which simplified the mechanical design and avoided a more complicated belt drive solution. We selected a dual motor controller so both motors could be controlled with one board, while still meeting the per-channel current requirement and keeping wiring and packaging simpler. To ensure that the Motor Drive Subsystem is fulfilling its responsibilities for controlled motor actuation, torque and speed capability, and safe operation, a requirements and verification table is provided below.

Table 2: Motor Drive Subsystem – Requirements & Verification

Requirements	**Verification**
• The motor drive subsystem must receive commands from the MCU and drive both the drive wheel motor and flywheel motor in the commanded direction and speed during normal operation. • The drive wheel encoder must provide usable speed feedback to the control subsystem.	• Ensure the robot is powered on and the MCU is running. Command low and moderate speed values to each motor. Confirm each motor spins in the expected direction. Command a direction change. Confirm the motor direction changes accordingly. • Record the drive wheel encoder reading while the wheel is still, then while rotating at low speed and higher speed. Confirm the reading changes in the correct direction and increase with wheel speed.
• The motor drive subsystem must maintain controlled operation during a representative test condition and must not cause loss of motor control, controller shutdown, or unexpected reset while both motors are active.	• Run the robot in a representative condition with the drive motor and flywheel motor active at the same time under MCU control. • Observe the system during startup, speed change, and steady operation. Confirm the motors continue responding to commands and no controller fault or shutdown occurs. Record the test condition and mark pass or fail.
• The drive wheel motor path must provide enough torque for the robot to recover from a tilted condition up to 45 degrees and pivot the	• Place the robot in a controlled setup at several tilt angles up to 45 degrees. Command the recovery action and confirm the robot

robot back toward upright.	pivots back toward upright. Record which angles pass.
• The motor drive subsystem must transmit power through the belt and pulley assembly and the flywheel hub interface without slipping during normal operation.	• During the same test and during normal motor operation, observe the belt and pulley assembly, shaft interfaces, and flywheel hub connection. Confirm there is no visible slipping, loosening, or loss of power transmission.
• The motor drive subsystem must remain thermally safe during continuous operation, and motor surface temperature must remain below 60 C during the defined test condition.	• Run the robot in a representative continuous operating condition for the team-selected test interval. Measure and record the surface temperature of each motor at the end of the test. Confirm each motor is below 60 C. Confirm the motor controller does not enter thermal shutdown during the test.

2.2.3 Sensing Subsystem

The Sensing Subsystem is responsible for providing the motion and orientation data used by the control subsystem to drive the motors properly. In the current design, this mainly includes the IMU and encoder feedback from the selected motors. This data is sent to the control subsystem and used in the feedback loop for balancing and motion control. Compared to an earlier version of the design, encoder sensing is now simpler because encoder feedback is built into the motor path instead of using a separate external encoder assembly. Even with that change, the purpose of this subsystem is the same. It must provide reliable information about robot orientation, wheel motion, and direction so the controller can make stable motor commands.

Since the control subsystem uses 3.3V logic, the sensing interfaces must also be compatible with the control electronics. The sensing subsystem must also remain reliable while the motors are active, since noisy or unstable sensor data would directly hurt feedback control performance. To ensure that the Sensing Subsystem is fulfilling its responsibilities for robot orientation and motion feedback, a requirements and verification table is provided below.

Table 3: Sensing Subsystem – Requirements & Verification

Requirements	Verification
• The sensing subsystem must provide IMU	• Ensure the robot is powered on and the

orientation data and encoder motion data to the control subsystem during normal operation. • The control subsystem must detect changes in these values when the robot body or drive wheel moves.	control subsystem is running. Record IMU and encoder values while the robot is still. • Then, rotate the robot body by hand and rotate the drive wheel. Record the values again. Confirm the IMU value changes with robot rotation and the encoder value changes with wheel motion.
• Encoder feedback must be usable for closed-loop control. The reported wheel speed must be near zero when the wheel is still and must increase when the wheel speed increases.	• Leave the drive wheel still and record the reported encoder speed. Confirm the value is near zero. • Then run the drive wheel at a low speed and record the encoder speed. Increase the speed and record again. Confirm the reported speed increases with wheel speed.
• IMU data must be stable enough for balancing control when the robot is stationary, and must report angle changes in the correct direction with usable resolution for control calculations.	• Leave the robot in a fixed position and record IMU angle values for a short interval. Confirm the value does not jump erratically while stationary. • Then rotate the robot by a small angle and record the IMU output. Confirm the reported angle changes in the correct direction and by a measurable amount.
• The sensing subsystem must operate with the control subsystem voltage level and remain readable during normal operation with motors active.	• Power the robot in a normal operating condition and confirm IMU and encoder data can be read by the control subsystem while idle. • Then run the motors and continue reading sensor values. Confirm sensor data remains available and does not drop out during the test.

2.2.4 Control Subsystem

The Control Subsystem is responsible for taking data from the sensing subsystem and turning it into motor commands that keep the robot balanced and moving the way we want. In other words, this is the part that closes the loop. The encoders and IMU report what the robot is doing right now, and the control code decides what the motors should do next so the robot returns to steady state or follows a user command.

Our control system is built around an ESP32 S3 module as the main real-time controller. It reads the IMU over a digital bus and reads the encoder signals from the drive motor, then outputs direction and PWM signals to the motor driver power control pins. The motor driver then routes the battery voltage to the motors based on those control signals. This separation is nice because the ESP32 only handles low-power logic-level signals, while the motor driver handles the high-current switching. The Raspberry Pi is mainly for computer vision and higher-level tasks, and it is not in the fast balancing loop.

We also want the control loop to be predictable. If the loop timing slips too much, the robot will feel inconsistent, and it can become harder to tune. So our requirements focus on loop rate, timing jitter, attitude accuracy, and disturbance recovery. The IMU choice supports fast reporting and a standard digital interface, and the ESP32 has hardware timers that let us run a periodic control task instead of relying on slow delay-based loops.

Table 4: Control Subsystem – Requirements & Verification

Requirements	**Verification**
• The control subsystem shall run the balancing control loop at 200 Hz or higher during normal operation, with timing jitter no greater than 1 ms per cycle.	• Load the control firmware and enable serial logging of loop timing. Run the controller for 60 s. Record loop period data. Confirm loop rate is at least 200 Hz and cycle timing jitter stays within 1 ms.
• The control subsystem shall read sensing data and generate motor control commands fast enough to support balancing, including tilt estimate resolution of 0.05 degrees, stationary tilt error within 1.0 degrees, and drift less than 2 degrees over 60 s.	• Secure the robot in a fixed setup and record the reported tilt angle over serial while stationary. Compare to a reference angle at several fixed tilt positions. Confirm the reported tilt changes in the correct direction, have usable resolution for control, stay within 1.0 degrees of the reference, and drifts less than 2 degrees over 60 s.
• The control subsystem shall read sensing data and compute PID control output fast enough to support balancing, including tilt estimate resolution of 0.05 degrees, a stationary tilt error within 1.0 degrees, and drift less than 2 degrees over 60 s.	• Run a motor control test program on the MCU. Command forward and reverse motion for each motor channel and vary the speed command. Confirm both channels respond correctly and the motor driver control inputs are driven by the MCU logic interface during

	the test.
• The PID control loop shall recover from a small disturbance by returning the robot tilt to within 2 degrees of its initial value within 2 s during the balancing operation.	• Put the robot in balancing test mode with a safe support setup. Apply a small repeatable disturbance and record tilt angle versus time from the serial output. Confirm the robot returns to within 2 degrees of the starting tilt within 2 s.
• If PID integral action is enabled, the control subsystem shall remain stable during actuator saturation, and a 0.5 s saturation event shall not increase settling time by more than 50 percent compared to the non-saturated case.	• Run and record a baseline PID step response test. Then repeat with a forced 0.5 s actuator saturation condition in software. Measure settling time for both tests from the logged tilt response. Confirm that the saturated case settling time increase is no more than 50 percent.

2.2.5 Compute Subsystem

The Compute Subsystem is responsible for higher-level processing tasks that are not part of the fast balancing loop. In our project, this mainly includes camera handling, video streaming, user-facing server functions, and computer vision processing for path and hazard detection. The compute subsystem also exchanges messages with the control subsystem so that vision results and user commands can be passed into the robot control flow. In our design, the compute subsystem is implemented on a Raspberry Pi. This is a practical choice because it already gives us onboard wireless connectivity, enough memory options for software development, and multiple I O interfaces for cameras and communication. Raspberry Pi 4 provides built-in Wi-Fi and Bluetooth, USB ports, a camera connector, and standard GPIO, which fits our compute and integration needs well.

The requirement for two cameras also fits this approach. We need one standard camera and one thermal camera. A common and practical setup is to use one camera through the Raspberry Pi camera interface and the other through USB, then process both streams in software on the Pi. Raspberry Pi documentation also supports the camera software stack and capture tools used for camera integration and streaming workflows.
The compute subsystem must also communicate with the control subsystem through a low-latency interface that is safe for both boards. Since the control subsystem is based on ESP32 S3 and uses 3.3V logic, the compute to control interface must also use 3.3V logic level signaling to avoid damaging pins. The ESP32 S3 module datasheet lists 3.0V to 3.6V as the recommended supply range and gives digital input and output thresholds based on VDD, which supports this 3.3V logic level interface requirement. To ensure that the Compute Subsystem is fulfilling its

responsibilities for camera input, wireless access, vision processing support, and communication with the control subsystem, a requirements and verification table is provided below.

Table 5: Compute Subsystem – Requirements & Verification

Requirements	**Verification**
• The compute subsystem must connect to two cameras, one standard camera and one thermal camera, and read image data from both during normal operation.	• Connect the standard camera and thermal camera to the compute subsystem in the intended configuration. Start the camera software and record one image or a short video stream from each camera. Confirm that both camera feeds are detected and readable.
• The compute subsystem must support user access and video streaming over a wireless connection during normal operation.	• Power on the compute subsystem and connect to it over the intended wireless interface. Start the local server and video stream. Confirm that a user device can connect and view the stream without losing connection during a short test run.
• The compute subsystem must exchange messages with the control subsystem through a low-latency 3.3V logic interface without communication failure during normal operation.	• Connect the compute subsystem to the control subsystem through the intended interface. Send a test message from compute to control and record the received value. Then send a reply message from control to compute and record the received value. Confirm both directions work correctly during repeated message exchange.
• The compute subsystem must run the intended computer vision program fast enough to support hazard and safe path identification during robot operation.	• Run the selected vision program on live camera input or recorded test footage. Record the processing rate and whether hazard or path outputs are produced. Confirm the system can process frames continuously and produce usable outputs during the test.

2.2.6 Vision Subsystem

The Vision Subsystem is responsible for providing raw image data to the compute subsystem. It acts as the image sensing input for the computer vision pipeline by capturing a live view of the robot's surroundings using a standard camera and a thermal camera.

This subsystem is separate from the compute subsystem. The vision subsystem focuses on camera hardware, image feed availability, and image delivery to the compute subsystem. The compute subsystem then handles processing, streaming, and higher-level computer vision tasks such as path and hazard detection. To ensure that the Vision Subsystem is fulfilling its responsibilities for camera sensing and image feed delivery, a requirements and verification table is provided below.

Table 6: Vision Subsystem – Requirements & Verification

Requirements	**Verification**
• The vision subsystem must provide both a standard image feed and a thermal image feed to the compute subsystem during normal operation. • The standard camera feed must support at least 1920 by 1080 resolution, and the thermal camera feed must support at least 192 by 192 resolution.	• Connect both cameras to the compute subsystem in the intended configuration. Open the camera feeds and confirm both are detected and streaming. • Record the reported resolution for each feed. Confirm the standard camera is at least 1920 by 1080 and the thermal camera is at least 192 by 192.
• The vision subsystem must deliver image feeds to the compute subsystem with low enough delay for real-time use and must remain available during normal operation without repeated disconnects.	• Run both camera feeds at the same time for a short continuous test. Confirm both feeds remain active during the test and do not repeatedly disconnect. • During the same test, move an object in front of each camera and observe the display or recorded output. Confirm the image updates fast enough for real time monitoring and hazard observation.
• The vision subsystem must remain mechanically and electrically stable during normal robot operation so that camera position and camera connection are maintained.	• Mount both cameras in the intended robot configuration and run the robot in a representative motion test. Confirm the camera mounts remain secure and the camera connectors do not come loose during the test. Confirm both image feeds remain available after the motion test.

2.3 Tolerance Analysis

One of the bigger concerns with our design is providing enough overall mechanical force in order to have the robot be able to balance itself, especially under conditions where the robot is off-center. This introduces a general torque requirement for our motor drive subsystem, depending on the specific constraints we set for our robot. For our current design, we have set the following physical requirements:

- Must weigh at most 2.0kg
- Must be at most 0.2m tall
- Center of mass must be at the center of the robot, or lower
- Max tilt angle of 45 °

Given these constraints, it is possible to derive the total amount of torque required in order to keep the robot upright from one of the motor drive subsystems.

$$\tau = r \cdot F = (\tfrac{0.2}{2})(2.0 * 9.81 * sin(45)) = 1.669\, Nm \qquad (1)$$

Given that this is assuming best case performance of our motors and assuming no mechanical losses, we need to add a safety factor to absolutely ensure that the robot will be capable of pivoting itself. We select a safety factor of 2 to guarantee robot functionality, without overloading the motors. This leaves us with a final required torque value of $3.338Nm$ for pivoting about a single axis and provides a baseline for the selection of components within our motor subsystem.

Looking around, there are motors that, when paired together, are capable of providing the torque we require, as well as driver boards that are capable of driving these without getting overloaded or being cost-prohibitive. An example motor is the FIT0186 gearmotor [2], capable of delivering $1.8Nm$ torque at stall individually. Two of these could be paired together to obtain a total of $3.4Nm$ of total torque at stall, well above our required torque with our safety factor included. These draw a maximum current of 7A each, and an example motor controller that could drive these is the DFR0601 dual-channel motor driver [3], capable of providing up to 12A of continuous current on each channel. Therefore, while the general torque requirement remains a concern within our robot, it is possible to meet these mechanical requirements as long as specific constraints are imposed on the physical design of the robot itself.

2.4 Physical Design

The physical design was one of the hardest parts of this project because the robot is small, the mass budget is tight, and many parts that look cheap at first become expensive or hard to integrate once shaft interfaces and mounting details are considered. Our outline constraints are a maximum mass of about 3 kg and a maximum size of about 30 cm, so packaging and part compatibility matter just as much as raw motor specs.

A lot of early ideas used more generic mechanical parts such as steel rod stock, loose bearings, and low cost belts and pulleys from Amazon. In practice this kept creating roadblocks. The main problem was not just price. It was finding parts that actually interface correctly with the selected motor shafts and can be assembled into a reliable drivetrain without too much trial and error. Timing pulleys with the right bore were hard to source, some generic parts were unexpectedly expensive, and trying to mix vendors made the design process slow before the CAD was fully settled.

Because of that, we decided to commit to a goBILDA based mechanical ecosystem for most of the drivetrain and flywheel hardware. This increases cost on some small parts such as REX bearings and shafts, but it greatly reduces fitment risk and makes assembly more straightforward. The shafts, hubs, pulleys, collars, and bearings are all designed to work together, which is important for a first full system build where we are also designing custom printed chassis parts.

The drive wheel assembly is built around an 8 mm REX shaft supported by flanged REX bearings mounted to the chassis. A Rhino wheel is mounted to the shaft through an 8 mm REX polycarbonate hub because the wheel does not directly interface to the shaft. A 24 tooth 8 mm REX bore timing pulley is also mounted on the same shaft. The selected FIT0186 drive motor uses a 6 mm D shaft, so a 16 tooth 6 mm D bore timing pulley is mounted on the motor shaft and connected to the wheel shaft pulley using a timing belt. Clamping collars are used to keep the shaft components in place and prevent axial shifting during operation. This gives a simple and buildable belt reduction stage and keeps the wheel shaft fully supported by bearings instead of loading the motor shaft directly.

The flywheel assembly is simpler than the original concept because we moved away from a separate belt driven flywheel. The current design uses a goBILDA 5000 series motor with an 8 mm REX output shaft and a steel flywheel mounted through a 1313 Series REX Hyper Hub. This direct drive layout removes extra pulleys, extra bearings, and an additional shaft stage. It also makes packaging easier and reduces the number of parts that can loosen during testing.

The printed chassis and frame hardware tie these assemblies together and support the battery, control electronics, and compute electronics near the center of the robot. PETG filament, M3 hardware, M3 threaded inserts, and M4 fasteners are included in the BOM for this purpose. This printed structure is doing more than just holding parts. It also sets shaft alignment, motor mount position, bearing spacing, and access to connectors and fasteners for assembly and service.

Figures 3 and 4 below should be referenced to show the overall packaging layout, including the drive wheel assembly support geometry, the main chassis width, and the placement of the flywheel structure relative to the body. This figure is useful for showing how the robot stays within the size target while still fitting the wheel shaft supports and electronics volume.

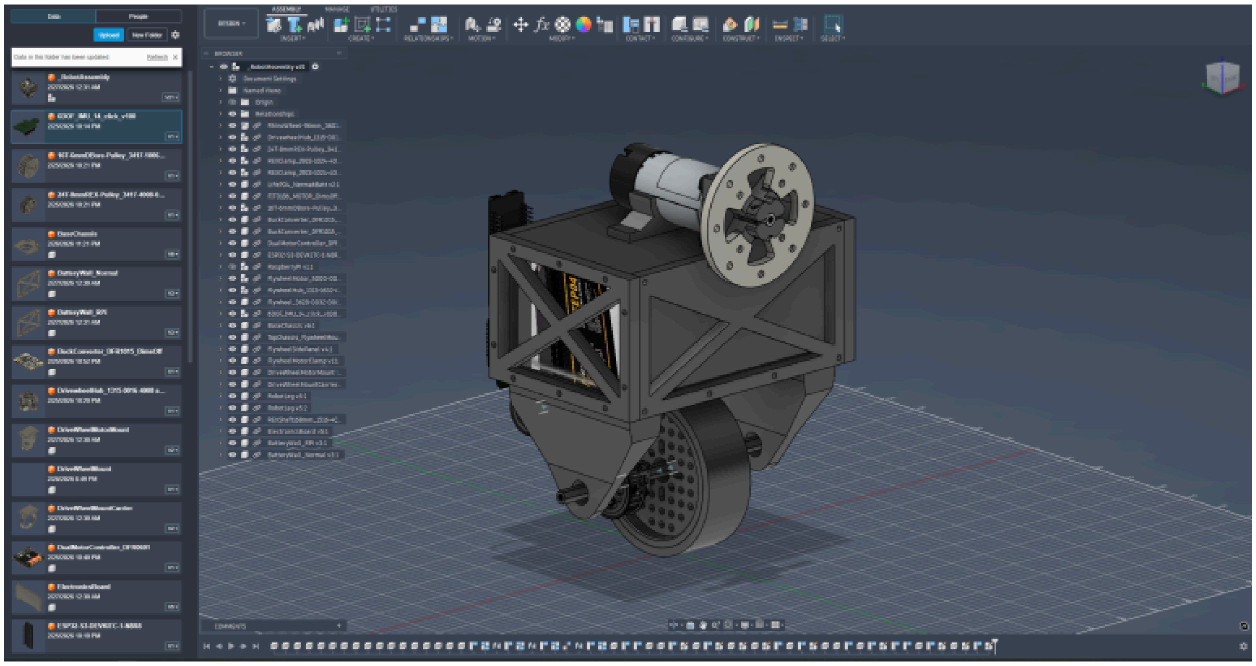

Figure 3. Front View of Mechanical Cad Design

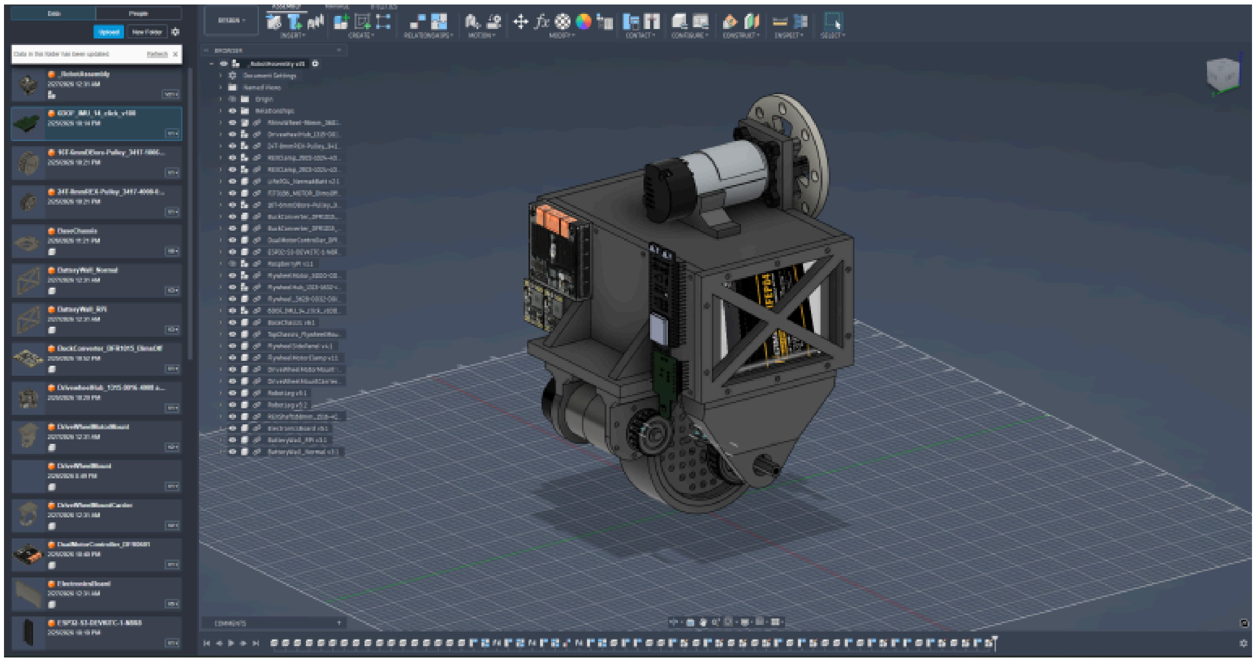
Figure 4. Back View of Mechanical Cad Design

2.5 Hardware Design

The hardware design is organized around a high current motor power path and a lower power logic and sensing path. This split is intentional. The motors need direct battery power through a motor driver, while the control and compute electronics need regulated low voltage supplies and low noise signal paths.

The main power source is a LiFePO4 battery selected for a practical balance of safety, current capability, cost, and weight. This battery chemistry was chosen instead of NiMH and LiPo after comparing current output, charging safety, and the total effort needed to use the battery safely in a student lab workflow. The battery path uses XT60 connectors for a robust high current connection and quick disconnect behavior during testing. A fuse holder and fuse are placed in the battery supply path for basic overcurrent protection.

Battery voltage is routed directly to the motor driver power input so the motor driver can supply the drive motor and flywheel motor from the battery source. In the current design, the selected motor driver is the DFR0601 dual channel driver. This board was chosen because one board can drive both motors, which keeps the wiring cleaner and reduces footprint compared to using two separate driver boards. It also matches the selected motor current range well and includes built in protection features and heat sinking, which is useful during iterative testing.

The low voltage electronics supplies are generated using two DFR1015 buck converter modules. These modules are used to generate the 5V and 3.3V supplies needed by the compute and control electronics. The 5V supply is intended for the Raspberry Pi and other 5V loads. The 3.3V supply is used for the ESP32 based control electronics and sensing interfaces. The buck modules were selected because they are not excessively oversized for this project, can be configured to the required output voltages, and include overload and overtemperature protection. This makes them a practical off the shelf choice and avoids adding a more complicated custom switching regulator design to the PCB at this stage.

The control hardware is centered on the ESP32 S3 platform. The ESP32 reads the IMU and encoder data, runs the fast control loop, and sends command signals to the motor driver. The key hardware point here is that the MCU handles only low power control signaling while the motor driver handles the high current switching. This separation reduces risk to the MCU and makes the control PCB design simpler. The selected IMU module is the MIKROE 4237 board based on the ICM 42688 device. This was chosen because the project needs stable and responsive motion sensing for balancing, and this module was available through the parts pipeline the team is already using.

A resistor divider is included in the power hardware for battery voltage monitoring. This is an important design feature because it allows the controller to estimate battery level during operation and determine when the pack should be recharged. The divider is not only for debugging. It supports normal operation and battery protection decisions in software by giving the control system a scaled battery voltage signal that is safe for the controller input.

The compute hardware is separated from the real time control path and is handled by a Raspberry Pi used for camera handling, streaming, and computer vision tasks. The hardware interface between the compute subsystem and control subsystem must remain at 3.3V logic level to avoid damaging GPIO pins. This requirement affects how the inter board communication lines are wired and is part of why the control and compute roles are separated into two boards instead of forcing all tasks onto one controller.

In summary, the hardware design choices here prioritize buildability and safe integration over maximum custom hardware complexity. The design uses a protected battery path, a dedicated motor driver for high current switching, off the shelf buck converters for regulated supplies, an ESP32 S3 for real time control, and a Raspberry Pi for higher level compute tasks. This gives a clear hardware split that is easier to debug and more practical to assemble within the project schedule.

2.6 Cost Analysis

Description	Manufacturer	Quantity	Extended Price	Link (URL)
Power Subsystem				
LiFePO4 Battery (12V 10Ah)	Weize	1	$33.99	https://www.amazon.com/dp/B097BRKCQP
LiFePO4 Battery Charger	NOCO	1	$29.95	https://www.amazon.com/dp/B07W46BX31
DC-DC Buck Converter	DFRobot	2	$5.30	https://www.digikey.com/en/products/detail/dfrobot/DFR1015/18069278
XT60 Connectors	DFRobot	3	$1.50	https://www.digikey.com/en/products/detail/dfrobot/FIT0587/9559256
Fuse Holder	—	1	$0.00 (ECE Lab)	ECE Self-Service Shop
Fuses	—	3	$0.00 (ECE Lab)	ECE Self-Service Shop
Capacitors	—	1	$0.00 (ECE Lab)	ECE Self-Service Shop
Motor Drive Subsystem				
Gearmotor w/ Encoder (Drive Wheel)	DFRobot	1	$16.50	https://www.digikey.com/en/products/detail/dfrobot/FIT0186/6588528
5000 Series Brushed Motor (Flywheel)	goBILDA	1	$29.99	https://www.gobilda.com/5000-series-12vdc-motor-with-8mm-rex-pinion-shaft/
Dual Channel Motor Driver	DFRobot	1	$29.00	https://www.digikey.com/en/products/detail/dfrobot/DFR0601/10279757
Sensing & Control Subsystem				
IMU Breakout Board (ICM-42688)	Mikroe	1	$30.00	https://www.digikey.com/en/products/detail/mikroelektronika/MIKROE-4237/13166617
ESP32-S3-WROOM-1 Module	Espressif	1	$5.06	https://www.digikey.com/en/products/detail/espressif-systems/ESP32-S3-WROOM-1-N4/16162639
Compute & Vision Subsystem				
Raspberry Pi 4	Raspberry Pi	1	$139.00	https://www.amazon.com/raspberry-pi-4/s?k=raspberry+pi+4

Standard Camera (1080p)	TBD	1	$29.99	https://www.amazon.com/Arducam-Camera-Raspberry-Windows-Android/dp/B07YHK63DS?th=1
Thermal Camera	TBD	1	TBD	TBD
Mechanical Parts				
Rhino Wheel	goBILDA	1	$8.99	https://www.gobilda.com/3601-series-rhino-wheel-14mm-bore-96mm-diameter/
REX Shaft 8mm	goBILDA	1	$5.99	https://www.gobilda.com/8mm-rex-shaft-with-e-clip-stainless-steel-120mm-length/
16T Belt Pulley (6mm D-bore)	goBILDA	1	$7.99	https://www.gobilda.com/2mm-pitch-gt2-pinion-timing-pulley-6mm-bore-20-tooth/
24T Belt Pulley (8mm REX bore)	goBILDA	1	$8.99	https://www.gobilda.com/2mm-pitch-gt2-pinion-timing-pulley-8mm-rex-bore-30-tooth/
Flanged REX Bearings 8mm	goBILDA	2	$3.99	https://www.gobilda.com/1611-series-flanged-ball-bearing-6mm-id-x-14mm-od-5mm-thickness-2-pack/
REX Clamping Collars	goBILDA	4	$19.96	https://www.gobilda.com/2910-series-aluminum-clamping-collar-8mm-rex-id-x-20mm-od-9mm-length/
8mm Polycarbonate Hub	goBILDA	2	$7.99	https://www.gobilda.com/1315-series-polycarbonate-hyper-hub-8mm-rex-bore-2-pack/
Timing Belt (60T)	goBILDA	1	$7.99	https://www.gobilda.com/2mm-gt2-pulleys/
Steel Flywheel (82mm Diameter)	goBILDA	1	$15.99	https://www.gobilda.com/steel-flywheel-82mm-diameter-152g-1651-g-cm/
1313 Series REX Hyper Hub	goBILDA	1	$12.99	https://www.gobilda.com/1313-series-hyper-hub-8mm-rex-bore/
PLA Filament (1kg)	HATCHBOX	1	$28.00	https://www.amazon.com/s?k=hatchbox+pla+filament+1kg
M3/M4 Screw Assortment	—	1	$0.00 (ECE Lab)	ECE Self-Service Shop
Wire & Connectors (misc)	—	1	$0.00 (ECE Lab)	ECE Self-Service Shop / Team Stock
Parts Total			**$449.16**	

The total cost for parts as seen below before shipping is ~ \$450. 5% shipping cost adds another \$22.5 and 10% sales tax adds another \$50. We can expect a salary of \$45/hr×2.5 hr×60 = \$6750 per team member. We need to multiply this amount with the number of team members, \$6750 × 3 = \$20,250 in labor cost. This comes out to be a total cost of \$20,767.5.

2.7 Schedule

Week	Task	Person
Week 1 – 2 Feb 24 – Mar 7	Finalize subsystem block diagram & signal flow	Everyone
	Define PCB requirements for each subsystem	Everyone
	Research & select remaining parts; finalize BOM	Everyone
	Begin schematic capture in KiCad (power + motor driver)	Varun
	Begin CAD model of chassis and frame in Fusion 360	James & Varun
	Set up Git repo, ESP32-S3 dev environment & toolchain	Varun
Week 3 – 4 Mar 7 – Mar 21	Complete full schematic capture for all subsystems	Varun
	PCB layout — power subsystem & motor driver board	Varun
	CAD: finalize frame geometry, shaft & bearing placements	James & Varun
	3D-print v0 bracket mockups for fit-check	James & Varun
	ESP32-S3 bringup: UART, I²C (IMU), PWM for motor driver	Varun
	Research PID balancing algorithm; begin simulation	Zhanshuo
Week 5 – 6 Mar 21 – Apr 4	PCB layout complete — route all boards, run DRC	Varun
	PCB ORDER SUBMITTED TO FAB	Everyone
	All mechanical parts ordered (goBILDA, DigiKey)	Everyone
	PARTS ORDER PLACED — components in transit	Everyone
	IMU data pipeline working on devboard (ICM-42688-P)	Varun
	PID simulation complete; gains estimated from model	Zhanshuo
Week 7 – 8 Apr 4 – Apr 18	PCBs arrive — solder & bring up power subsystem	Zhanshuo & James
	PCBs arrive — solder & bring up motor driver board	Zhanshuo & James
	Assemble v1 mechanical prototype (frame + wheel + shaft)	James & Varun
	V1 PROTOTYPE MECHANICALLY ASSEMBLED	James & Varun

	Motor bench test: open-loop RPM, encoder, current draw	Varun
	MOTORS TESTED — validated against spec	Varun
	Rudimentary self-balancing control loop running on bench	Zhanshuo
	Closed-loop PID (angle only) — robot balances on stand	Zhanshuo
Week 9 – 10 Apr 18 – May 2	Iterate frame based on v1 test findings	James & Varun
	FINAL FRAME COMPLETE — approved for final assembly	James & Varun
	Fabricate or order any revised mechanical parts	Varun
	Confirm all PCBs functional with final assembly boards	Varun
	ALL PCBs VALIDATED on final boards	Varun
	Integrate PCBs into finalized frame; cable management	Zhanshuo & Varun
	Tune PID gains with robot free-standing	Zhanshuo
	Begin CV pipeline: camera capture & frame preprocessing	James
Week 11 – 12 May 2 – May 16	Full system integration — all electronics in final frame	Everyone
	FULLY FUNCTIONAL ROBOT — free-standing balance demo	Everyone
	PID gain scheduling and tuning for load variations	Zhanshuo
	Rail-tracking mode: camera + IMU sensor fusion test	Zhanshuo
	CV module: edge / rail detection algorithm v1 on SBC	James
	Begin CV software integration into control loop	James & Zhanshuo
	Mechanical stress test & vibration assessment	James & Varun
	Safety analysis: fault detection, E-stop, overcurrent	James
Week 13 – 14 May 16 – May 30	Full HW + SW integration: CV steering + balancing loop	Everyone
	ALL SUBSYSTEMS INTEGRATED & FUNCTIONAL	Everyone
	End-to-end demo run on simulated rail / narrow beam	Everyone
	Final robot tuning: speed, stability, CV responsiveness	Zhanshuo
	Safety verification: edge cases & fault injection tests	James
	Final hardware validation & robustness testing	Varun
	DEMO-READY — final verification complete	Everyone
	Prepare demo materials & final design document updates	Everyone

3. Ethics, Safety, and Societal Impact

With this project, we have a responsibility to acknowledge certain ethical responsibilities, ensure the safety of others, and address the potential societal impacts this robot may have. Especially in the modern age of consumer electronics, there have been increasing concerns surrounding users' privacy, the net benefit these products have in the long term, and ensuring that our project is safe to use both by its users and those who may be exposed to it as well. We have identified and addressed three main points of concern and how it has influenced the design of the robot to be both ethical and safe.

3.1 Privacy

Current-day smart devices often come with a camera built in, usually to provide a feature to users that enhances their experience with the product. For example, smart doorbells may have a camera built in so that a homeowner is able to view who is at their door without having to be present at the door or even inside their own house. However, these devices with cameras are cause for privacy issues, as these companies may be allowed to take footage recorded from these smart cameras and use them to learn more about the user, their habits, and other personal information that users are unaware of. With the design of Railrider, we want to prioritize privacy so that user privacy is protected as discussed in the IEEE Code of Ethics [1], and so that no unnecessary data is collected other than to observe the environment the robot is located in. All computer vision processing is done locally on the robot itself in order to ensure recorded video remains local to the user, and any live footage viewed is done on a closed, secure wireless network that only authorized users will have access to. This maximizes user privacy while allowing for Railrider to make use of advanced features, at the expense of needing to have enough compute be locally available on the robot for it to effectively work.

3.2 Safety

While working on and developing Railrider, we have to be mindful of both the mechanical and electrical dangers associated with this project. Two main hazards exist: the flywheel and the battery being used to power the robot.

As the flywheel will be spun to thousands of RPM in order to act as an energy storage device to help balance the robot, this comes with the risk of the flywheel undergoing complete mechanical failure at high speeds, sending hazardous shrapnel towards nearby individuals. In order to mitigate the risk of people getting hurt while the flywheel is undergoing testing, we aim to operate the flywheel in a safe, enclosed environment until the safety of the wheel has been mathematically guaranteed, and to also ensure the final flywheel will be constructed out of a robust material that is able to withstand being spun at high speeds. We will also make sure the flywheel is manufactured precisely to ensure balance and minimize vibrations that would lead to mechanical failure.

With respect to the LiPo battery we plan to use for our project, we plan to follow safe practices when charging batteries, such as keeping batteries charging within a battery bag when charging, and keeping our battery stored in a safe, non-flammable location in case of failure. We include a BMS within our project in order to ensure that all battery cells are balanced and not reaching dangerously high or low voltages, and aim to only begin using such a battery within our project only when we know the general operation of our robot is achievable on a bench power supply. We also aim to add a checklist for the robot whenever we plan to begin using a battery with our system, checking wiring, boards, and contacts around the robot to guarantee it is safe to plug the battery in before doing so. This thereby mitigates the amount of danger introduced to ourselves and those around during the testing and development process.

3.3 Public Well-Being

As technology progresses, there comes the fear of new machines and innovations rendering certain jobs obsolete. For instance, manufacturing throughout history has gone from having an assembly line of human workers to instead now having machines that are capable of assembling a car from beginning to end with little human intervention. With this in mind, there is reason for concern that a project such as Railrider may pave the way for certain human inspection jobs to become obsolete or useless. We wish not for our robot to fully automate the human inspection process of certain environments, but rather for it to act as another tool people can use in unideal conditions - tight, narrow spaces, thin-beamed structures, areas where human deployment is not feasible, etc. The compute capability we aim to deploy on the robot, while we aim to make it capable and performant, will still not fully replace the experience and knowledge a human has when identifying environmental hazards. The robot will still rely on a human operator to help use Railrider as a tool, and we hope that this enhances the job of inspectors and allows them to explore more areas that previously were not traversable. Thus, we aim to improve the quality of life for users of the product, rather than hindering it through innovation.

References

[1] IEEE, “IEEE Code of Ethics | IEEE,” *Ieee.org*, 2020.
https://www.ieee.org/about/corporate/governance/p7-8
[2] FIT0186 Dfrobot | motors, actuators, solenoids and drivers | digikey,
https://www.digikey.com/en/products/detail/dfrobot/FIT0186/6588528 (accessed Feb. 14, 2026).
[3] DFR0601 dfrobot | motors, actuators, solenoids and drivers | digikey,
https://www.digikey.com/en/products/detail/dfrobot/DFR0601/10279757 (accessed Feb. 14, 2026).