

Self-Balancing Bike ISR Report

Aryan Banerjee

Abstract

This report documents my technical contributions to and independent learning through the Autonomous Self-Balancing Bicycle (ASB) project during Spring 2026. The bicycle is a custom-instrumented Retrospec Harper fixed-gear frame modified to achieve autonomous dynamic balance and low-speed navigation. My contributions spanned electrical architecture, power design, wiring, and sensor integration, with a particular focus on the STM32-based balancing controller, the reaction wheel drive system, and IMU integration. The reaction wheel, a water-jet cut and CNC-machined low-carbon steel disk driven by an ODrive M8325s motor through an ODESC V4.2 controller, serves as the primary actuation mechanism for balance. A PID control loop reads estimates from an ICM-45686 IMU mounted on anti-vibration standoffs and commands corrective torque to the ODESC V4.2 controller over UART to maintain upright stability. Through iterative tuning involving P-only isolation and Ziegler-Nichols methods, the system achieved consistent balancing while stationary. The intended navigation stack centers on an NVIDIA Jetson Orin Nano as the onboard compute unit, a ZED 2i stereo camera as the primary sensor for visual SLAM, and a 2D LiDAR for obstacle detection. Additionally, as an independent learning objective for this course, I studied Simultaneous Localization and Mapping in depth, focusing on ORB-SLAM3, a well-regarded feature-based visual-inertial SLAM system, and developed a hypothetical architecture describing how it could be integrated with the bicycle to support autonomous navigation. This report reflects both the engineering work I contributed to and the technical understanding I developed independently.

Introduction

Autonomous bicycles occupy a compelling and underexplored niche in mobile robotics. Unlike four-wheeled ground vehicles, a bicycle is dynamically unstable by nature: without active intervention, it falls. This instability is not incidental but structural. A bicycle at rest or at low speed behaves as an inverted pendulum, a classical control problem in which the equilibrium point is inherently unstable and any small perturbation, if left uncorrected, grows exponentially. Bringing such a system under reliable closed-loop control requires fast sensing, accurate state estimation, and well-tuned actuation. At the same time, the bicycle's mechanical simplicity makes it an interesting candidate for autonomous navigation in environments where larger vehicles are impractical.

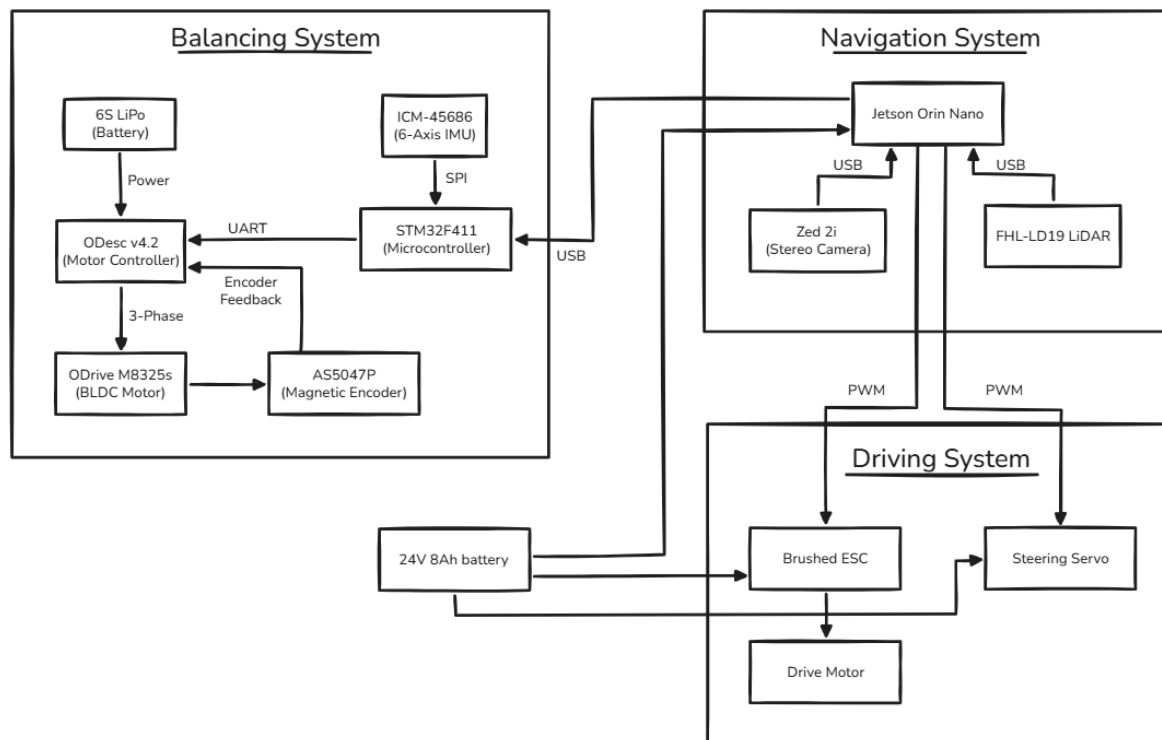
The Autonomous Self-Balancing Bicycle (ASB) project set out to address these challenges on a practical, hardware-first basis. The bicycle is built on a Retrospec Harper fixed-gear frame, modified to support motorized steering, a reaction wheel balancing mechanism, and a brushed DC drive system. I worked on the electrical architecture, power design, wiring, and sensor integration, with a particular focus on the STM32-based balancing controller, including the IMU interface, PID implementation, and tuning process. I also personally designed several custom mechanical mounts.

The balancing system operates by commanding corrective torques through a reaction wheel, which does not require the bicycle to be moving. This is a meaningful distinction from velocity-dependent stabilization methods such as steering-based control, which rely on forward motion to generate gyroscopic and caster effects. The reaction wheel approach treats the problem directly as an inverted pendulum: the wheel accelerates or decelerates to generate angular momentum that counteracts the bicycle's tendency to tip. During my time on the project, the system achieved consistent balancing while stationary.

The intended navigation stack builds on top of the balancing system. An NVIDIA Jetson Orin Nano serves as the onboard compute unit, responsible for running the navigation software and coordinating the drive and steering subsystems. A ZED 2i stereo camera provides the primary visual input for localization and mapping, while a 2D LiDAR sensor handles obstacle detection in the bicycle's immediate path. Together these components form the hardware basis for the autonomous navigation architecture explored in the second half of this report.

The second half covers Simultaneous Localization and Mapping as an independent learning objective for this course. SLAM addresses the problem of building a map of an unknown environment while simultaneously tracking position within it, a capability that any autonomous vehicle ultimately requires. I worked through the theory behind ORB-SLAM3, a well-regarded feature-based visual-inertial SLAM system, and developed a hypothetical architecture describing how it could be integrated with the bicycle to support navigation. The analysis is grounded in the real hardware constraints of the system as I built it.

System Overview



The ASB bicycle is composed of three largely independent subsystems: the balancing system, the drive and steering system, and the intended navigation stack. Each operates with its own dedicated hardware, and the boundaries between them are intentional. The balancing system was designed to be fully isolated, ensuring that instability or failure in any other part of the bicycle would not interfere with the core task of keeping it upright.

The balancing system centers on the STM32F411 microcontroller, which runs the PID control loop. It reads data directly from the ICM-45686 IMU over SPI and commands corrective torques to the ODESC V4.2 motor controller over UART. The ODESC drives the ODrive M8325s brushless motor, which spins the reaction wheel to generate the angular momentum needed to counteract tipping. This entire loop, from IMU to STM32 to ODESC to motor, is self-contained and operates independently of everything else on the bicycle. Power for this subsystem comes from a dedicated 6S LiPo battery, further isolating it electrically from the rest of the system.

The drive and steering system handles forward motion and directional control. A brushed DC motor drives the rear wheel through a 25H chain and a 90T/11T sprocket reduction, controlled by a 110A bidirectional brushed ESC. Steering is handled by a 160 kg-cm 24V servo mounted above the top tube, connected to the fork through a 1:1 spur gear arrangement. In the intended final design, both the ESC and the servo receive

PWM signals from the Jetson Orin Nano. The remainder of the electrical system runs from a separate 24V 8Ah lithium battery.

The navigation stack is built around the NVIDIA Jetson Orin Nano, which serves as the onboard compute unit. A ZED 2i stereo camera provides the primary visual input, feeding into ORB-SLAM3 for localization and mapping. The ZED 2i also contains a built-in IMU, which ORB-SLAM3 uses for visual-inertial odometry. A 2D LiDAR sensor supplements this by detecting obstacles in the bicycle's immediate path. The Jetson translates navigation outputs into PWM commands for the drive ESC and steering servo. The navigation stack and the balancing system do not communicate directly; in a fully realized implementation, the path planner would need to account for the balance constraints of the bicycle when issuing steering and speed commands, a challenge discussed further in the SLAM and navigation section.

Mechanical Design

Frame Selection

The Retrospec Harper is a fixed-gear bicycle with a clean tubular steel frame. The tubular frame geometry provided convenient mounting surfaces along the top tube, down tube, and seat tube, which proved important as the number of components grew. Notably, the top tube runs parallel to the ground, which offered a natural flat reference plane for mounting electronics and provided a clean geometric zero reference for the IMU, simplifying the angle estimation in the balancing controller. Removing the original drivetrain left the rear wheel free to be driven by the custom chain and sprocket system described below.

Reaction Wheel

The reaction wheel is the most mechanically significant custom component on the bicycle. The wheel itself was cut from a low-carbon steel disk, 6 inches in diameter and 1 inch thick. It was CNC machined and water-jetted into a spoke-like geometry to remove mass from the interior while preserving rotational inertia at the rim, where it contributes most to the flywheel effect. The spoke geometry reduces overall mass to keep the bicycle lighter while retaining enough angular momentum capacity to generate meaningful corrective torques.

The reaction wheel is driven by an ODrive M8325s 100KV brushless motor, which provides a peak torque of 6.6 Nm. Motor feedback is provided by an AS5047P magnetic encoder, which the ODESC V4.2 controller uses for closed-loop torque control. The ODESC is a derivative of the open-source ODrive motor controller and supports field-oriented control, allowing precise and responsive torque commands. The entire reaction wheel assembly is mounted near the seat tube via a custom 3D printed

bracket, positioning the mass centrally along the bicycle's longitudinal axis to minimize its effect on the overall center of gravity.

Steering

Steering is actuated by a 160 kg-cm 24V digital servo mounted above the top tube. The servo connects to the fork through a 1:1 3D printed spur gear arrangement, which transfers rotation directly to the fork steerer tube without any mechanical advantage or reduction. This keeps the steering response linear and predictable, which is important for a system where the controller needs to issue precise angle commands. The servo's 270 degree range of motion is more than sufficient for the steering angles required at the speeds the bicycle is intended to operate.

Drive System

The main drive system uses a brushed DC motor connected to the rear wheel through a 25H chain and a 90T/11T sprocket reduction, giving an overall gear ratio of approximately 8.2:1. This reduction trades speed for torque, which is appropriate for a bicycle that needs to accelerate from rest and maintain low speeds under load. Braking is handled entirely through motor braking via the ESC rather than mechanically, keeping the system simple and avoiding the need for a separate braking actuator.

Fabrication and Mounts

A number of custom 3D printed mounts were designed and fabricated over the course of the project to attach electronics, motors, and sensors to the frame. Rather than relying on off-the-shelf clamps, dedicated mounts allowed each component to be positioned precisely and secured reliably.

Electrical Architecture

Power System

The electrical system is divided into two fully isolated power domains, each with its own battery. This isolation was a deliberate design decision to prevent the high current demands of the reaction wheel motor from affecting the rest of the system, and to ensure that the balancing subsystem could operate independently regardless of the state of the navigation or drive hardware.

The reaction wheel domain runs from a 6S LiPo battery at 22.2V. This battery feeds directly into the ODESC V4.2 motor controller, which drives the ODrive M8325s brushless motor. The high discharge rate of the 6S LiPo, rated at 150C, ensures it can handle the peak current demands of the reaction wheel during rapid corrective torques without voltage sag affecting controller behavior.

The main system domain runs from a 24V 8Ah lithium battery. Terminal blocks serve as the central distribution hub for this domain, branching power out to the steering servo, the brushed DC ESC, and a 24V to 12V DC-DC converter that feeds the Jetson Orin Nano. A single toggle switch with a protective cover sits in the main power line, allowing the entire 24V domain to be safely enabled or disabled. The STM32F411 is powered directly from the Jetson via USB, keeping its power supply clean and avoiding the need for a separate voltage regulator.

Balancing Controller Interface

The STM32F411 communicates with the ODESC V4.2 over UART. The ICM-45686 IMU connects to the STM32 over SPI. The IMU is mounted on M3 anti-vibration rubber standoffs, which mechanically decouple it from frame vibrations that would otherwise introduce noise into the attitude estimate. At a sampling rate of 400 Hz, even small amounts of vibration-induced noise can meaningfully degrade controller performance, making the standoffs an important detail rather than an afterthought.

Navigation and Drive Interface

The Jetson Orin Nano interfaces with the drive and steering subsystems through PWM signals. The brushed DC ESC and the steering servo both accept PWM commands, and in the intended final design the Jetson generates these directly from its GPIO.

The ZED 2i stereo camera and the 2D LiDAR both connect to the Jetson over USB, and both are compatible with the Jetson's ROS environment, which simplifies integration on the software side.

Hardware Iteration and Firmware Development

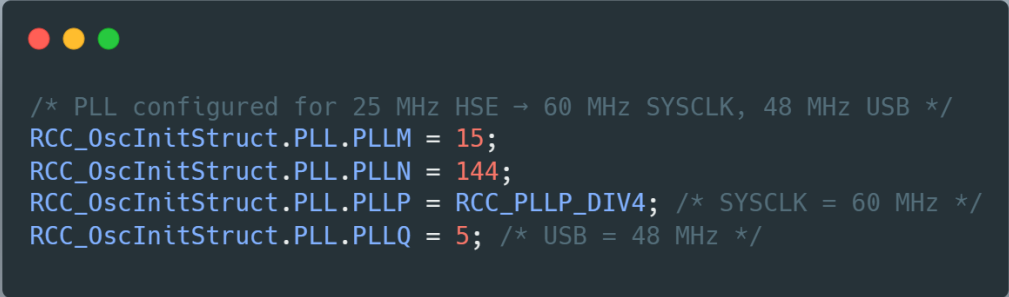
STM32 and CubeMX

The balancing controller runs on an STM32F411CEU6, mounted on a Black Pill development board. When selecting a microcontroller for this application, alternatives like the ESP32 were considered. However, the STM32 was the better fit for a hard real-time control loop: it offers dedicated hardware SPI peripherals, hardware timers, and a Cortex-M4 core that executes deterministically without the overhead of a wireless stack or the unpredictability that comes with a more application-oriented chip. For a 400 Hz balancing loop where the IMU must be read and a motor command issued within a fixed 2.5 ms window every cycle, that determinism matters. The Black Pill board is also compact and inexpensive, which made it practical to mount directly on the bicycle.

STM32CubeMX was used to configure the microcontroller peripherals. CubeMX is a graphical tool that generates HAL-based initialization code from a pin and peripheral configuration, which significantly reduces the time spent on boilerplate register setup. The key peripherals configured were SPI1 for the IMU, USART1 for the ODESC motor controller, and USB OTG FS in CDC mode for the debug terminal.

The clock configuration required particular attention. The Black Pill uses a 25 MHz external oscillator rather than the 8 MHz crystal found on most reference designs and online examples. This meant the standard PLL values found in tutorials were incorrect for this board. The PLL was configured with PLLM=15, PLLN=144, PLLP=DIV4, and PLLQ=5, producing a 60 MHz system clock. The PLLQ divider is specifically responsible for deriving the 48 MHz clock that USB requires, and this is where the first significant debugging challenge arose.

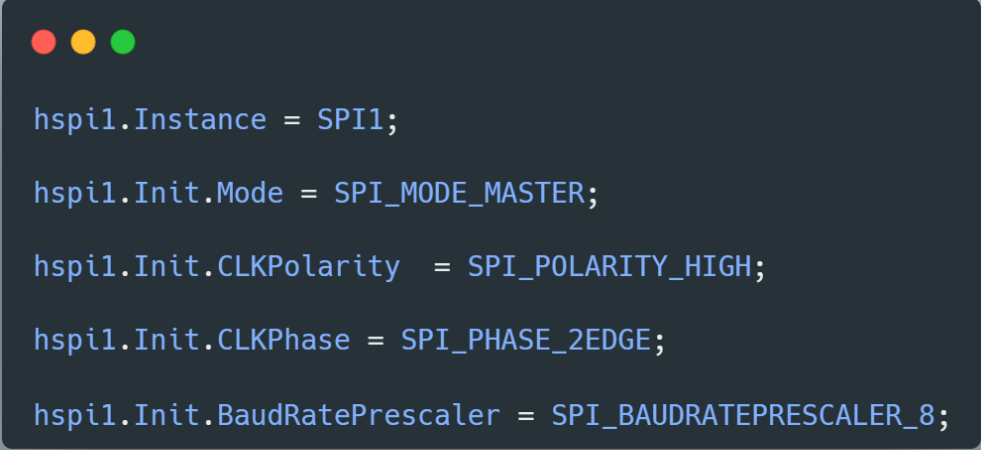
When USB CDC was first brought up, the device was failing to enumerate on the host computer, appearing as an unknown device. After investigation the issue traced back to the PLL configuration: because the board uses a 25 MHz oscillator rather than the standard 8 MHz, the PLLM and PLLN values from common references were producing a PLLQ output that was not exactly 48 MHz. USB enumeration is sensitive to this frequency and will fail if it is off. Correcting the PLL values in CubeMX and regenerating the initialization code resolved the issue. The final configuration is shown below:



```
/* PLL configured for 25 MHz HSE → 60 MHz SYSCLK, 48 MHz USB */  
RCC_OscInitStruct.PLL.PLLM = 15;  
RCC_OscInitStruct.PLL.PLLN = 144;  
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4; /* SYSCLK = 60 MHz */  
RCC_OscInitStruct.PLL.PLLQ = 5; /* USB = 48 MHz */
```

SPI1 was configured as a full-duplex master in Mode 3, meaning the clock idles high and data is sampled on the second edge, matching the ICM-45686's requirements. A prescaler of 8 gives a 7.5 Mbit/s SPI clock. SPI uses four lines: a clock line (SCK), a data line from master to peripheral (MOSI), a data line from peripheral to master (MISO), and a chip select line (CS) that goes low to begin a transaction and high to end it. Every transfer is full-duplex, meaning the master and peripheral exchange bytes simultaneously on every clock edge. Two settings that must match exactly between the master and peripheral are clock polarity (CPOL), which defines whether the clock idles high or low, and clock phase (CPHA), which defines whether data is sampled on the

first or second edge. Getting these wrong produces corrupted data with no obvious error, which made debugging initial communication issues more involved than expected. For the ICM-45686, the correct configuration is Mode 3, meaning CPOL=1 and CPHA=1: the clock idles high and data is sampled on the falling edge. The chip select line also required careful handling: it must be driven low before the first byte and held low for the entire transaction, then brought high again after the final byte. Releasing it early terminates the transfer and corrupts the message. CubeMX generated the SPI peripheral initialization but the transaction logic, including chip select control and register addressing, was written from scratch. The relevant CubeMX-generated configuration is:



```
hspi1.Instance = SPI1;

hspi1.Init.Mode = SPI_MODE_MASTER;

hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;

hspi1.Init.CLKPhase = SPI_PHASE_2EDGE;

hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
```

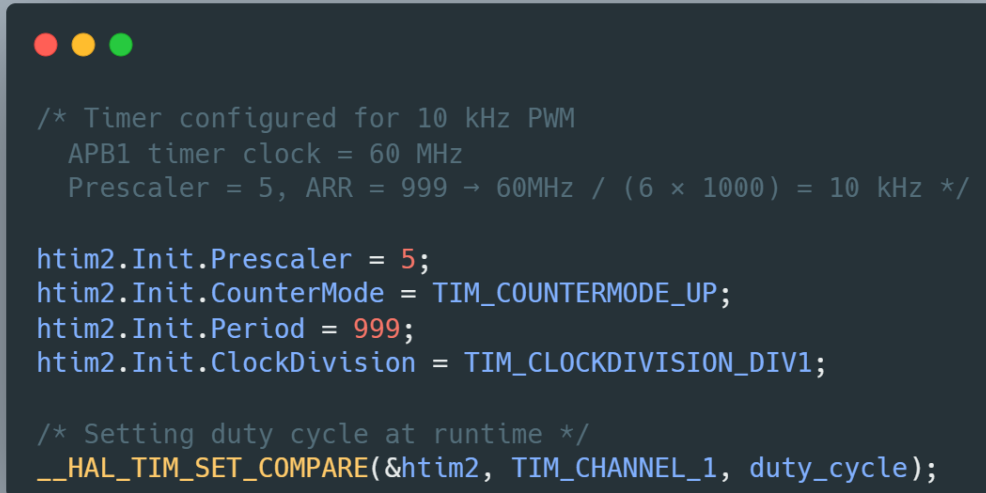
The firmware runs as a bare-metal superloop at 400 Hz. Each iteration reads the IMU, computes the filtered angle, checks for a fall condition, runs the PID, and sends a torque command to the motor controller. This architecture is simple and fully sufficient for a single-axis balance controller with predictable, fixed timing requirements.

Initial Motor and Controller Setup

The first reaction wheel motor was a NEMA 24 200W brushless DC motor rated at 3000 RPM and 24V, with a rated torque of approximately 0.65 Nm. This was paired with an XY-BLDC three-phase brushless controller. Getting this system running was the first real hardware bring-up task: generating a PWM signal from the STM32 to command the motor controller.

PWM is the standard way microcontrollers communicate speed commands to motor controllers, with the duty cycle interpreted as a throttle percentage. The XY-BLDC controller expected a signal at around 10 kHz. On the STM32, this is generated using a

hardware timer configured in PWM mode. The timer prescaler and auto-reload register set the frequency, while the compare register controls the duty cycle:



```
/* Timer configured for 10 kHz PWM
   APB1 timer clock = 60 MHz
   Prescaler = 5, ARR = 999 → 60MHz / (6 × 1000) = 10 kHz */

htim2.Init.Prescaler = 5;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 999;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;

/* Setting duty cycle at runtime */
__HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, duty_cycle);
```

Getting the frequency, duty cycle range, and signal levels correct involved a fair amount of iteration and built a strong understanding of how hardware timers work on the STM32.

Controller Limitation and Switch to ODESC

With the motor running, testing quickly revealed a fundamental problem with the XY-BLDC controller. It could not rapidly decelerate the motor or switch its direction of rotation without first coasting to a stop. For a reaction wheel balancing system this is a critical flaw. The reaction wheel must be able to reverse its angular momentum rapidly in response to the bicycle tipping in either direction, and any delay in that response means the bicycle continues to fall while the controller waits for the motor to spin down. This was observed directly in testing: the controller's inability to brake aggressively made sustained correction impossible, and the decision was made to switch to the ODESC V4.2.

The ODESC is a field-oriented control motor controller built on the ODrive open-source design. Unlike the XY-BLDC controller, it supports true bidirectional torque control with regenerative braking, meaning it can command the motor to actively decelerate and reverse without any coasting delay. The ODESC accepts commands over UART using a simple ASCII protocol, which the STM32 sends over USART1. With the ODESC in place the direction switching problem was resolved and the balancing loop had the responsive actuation it needed.

The Inverted Pendulum Model and Torque Requirements

Understanding why the original motor was insufficient requires first establishing the physics of what the reaction wheel is actually doing.

A bicycle at rest behaves as an inverted pendulum. If the bicycle has mass M , height to center of mass h , and leans at angle θ from vertical, the gravitational torque acting to tip it over is:

$$T_{gravity} = Mgh \times \sin(\theta)$$

For small angles $\sin(\theta)$ is approximately equal to θ , so this simplifies to:

$$T_{gravity} = Mgh\theta$$

This torque grows linearly with lean angle. The reaction wheel counteracts it by changing its own angular momentum. When the motor applies a torque to accelerate the reaction wheel, an equal and opposite torque is applied to the bicycle frame by Newton's third law. The torque the reaction wheel can exert on the bicycle is therefore:

$$T_{reaction} = I_{wheel} \times \alpha_{wheel}$$

where I_{wheel} is the moment of inertia of the reaction wheel and α_{wheel} is its angular acceleration. For the system to balance, $T_{reaction}$ must be able to match or exceed $T_{gravity}$ within the operating lean angle range.

Taking approximate values for the bicycle, with a total mass of around 15 kg and a center of mass height of around 0.5 m, at a lean angle of just 3 degrees or approximately 0.05 radians:

$$T_{gravity} = 15 \times 9.81 \times 0.5 \times 0.05 \approx 3.7 \text{ Nm}$$

The original NEMA 24 motor had a rated torque of 0.65 Nm, which falls far short of what is required even at a very small lean angle.

Motor Torque Limitation and Switch to ODrive M8325s

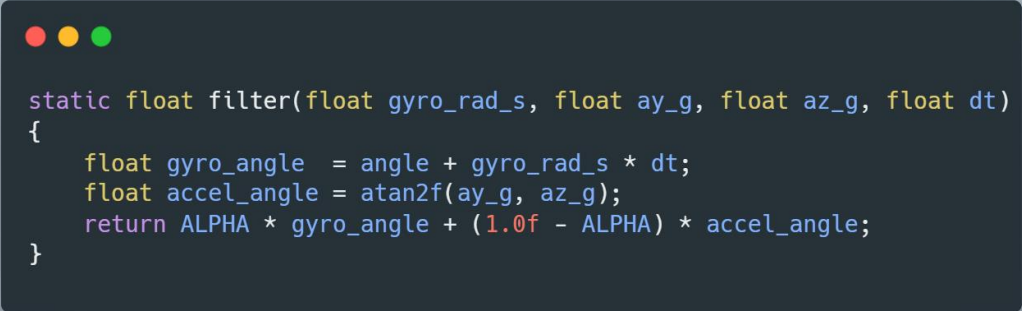
In practice this limitation showed up clearly. When the PID loop engaged with the original motor, the reaction wheel would spin up and for a brief moment push back against the lean, demonstrating that the control architecture was fundamentally sound. However the motor did not have enough torque to bring the bicycle back to the balance point, and the bicycle would continue to fall. This brief pushback was nonetheless useful: it confirmed that the approach was correct and acted as a proof of concept for the overall system, even if the hardware was not yet adequate.

The analysis above made it clear that the problem was not in the firmware or the control logic but in the motor's physical torque capacity. The ODrive M8325s was selected as the replacement, with a peak torque of 6.6 Nm, comfortably exceeding the gravitational torque at the lean angles where the balancing loop operates. This gave the controller enough authority to actually correct a fall rather than merely slow it down. The switch to the M8325s, combined with the ODESC controller already in place, was what made sustained balancing achievable. USB CDC telemetry throughout this entire process was invaluable for diagnosing each issue as it arose, providing real-time visibility into the tilt angle, gyro rate, and motor commands during every test run.

Balancing Controller

Angle Estimation

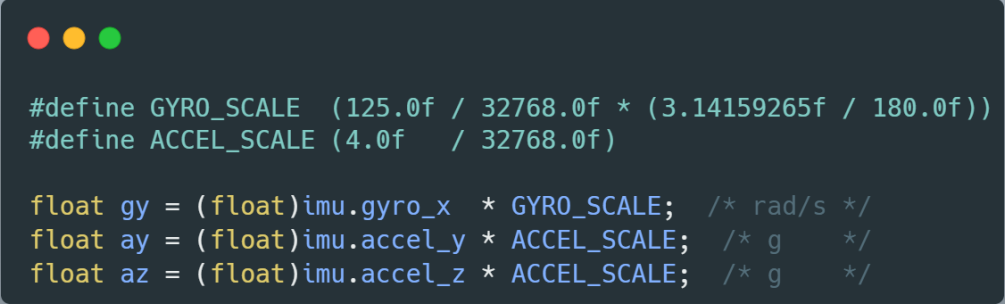
The balancing controller depends entirely on an accurate, real-time estimate of the bicycle's tilt angle. The ICM-45686 provides both accelerometer and gyroscope data at 400 Hz, and the angle estimate is derived by fusing both. Neither sensor alone is sufficient. The gyroscope measures angular rate directly and integrates cleanly over short time periods, but any small bias in the reading accumulates over time and causes the estimated angle to drift away from the true value. The accelerometer can estimate the static tilt angle from the direction of gravity, but it is sensitive to vibration and dynamic acceleration, making it noisy and unreliable during active motion. Combining the two at a heavily gyro-weighted ratio gives an estimate that is fast and accurate in the short term while being corrected for drift over longer periods:



```
static float filter(float gyro_rad_s, float ay_g, float az_g, float dt)
{
    float gyro_angle = angle + gyro_rad_s * dt;
    float accel_angle = atan2f(ay_g, az_g);
    return ALPHA * gyro_angle + (1.0f - ALPHA) * accel_angle;
}
```

With ALPHA set to 0.98, the gyroscope dominates the estimate on every tick while the accelerometer contributes a small corrective pull. At 400 Hz this correction accumulates meaningfully over time without introducing the accelerometer's vibration noise into the fast control loop. At startup the angle is seeded directly from the accelerometer to give the filter a sensible starting point rather than beginning from zero.

The ICM-45686 is configured with a gyro range of 125 dps and an accelerometer range of 4g, both at 400 Hz output data rate with a third-order low-pass filter at approximately 6 Hz bandwidth. The raw values are scaled in firmware before being passed to the filter:



```
#define GYRO_SCALE (125.0f / 32768.0f * (3.14159265f / 180.0f))
#define ACCEL_SCALE (4.0f / 32768.0f)

float gy = (float)imu.gyro_x * GYRO_SCALE; /* rad/s */
float ay = (float)imu.accel_y * ACCEL_SCALE; /* g */
float az = (float)imu.accel_z * ACCEL_SCALE; /* g */
```

Fall Detection

A fall detection check runs on every loop iteration before the PID is evaluated. If the absolute tilt angle exceeds 0.5 radians, approximately 29 degrees, the motor torque is set to zero and the PID integrator is reset. This threshold exists because beyond that angle the reaction wheel cannot physically generate enough torque to recover, and continuing to command the motor would only saturate it against a fall that is already unrecoverable. Resetting the integrator on recovery prevents the accumulated integral windup from causing a large torque spike when the bicycle is picked back up and the controller re-engages.

```
if (fabsf(angle) > FALL_THRESHOLD) {  
    ODrive_SetAxisTorque(0, 0.0f);  
    PID_Reset(&pid);  
    continue;  
}
```

PID Controller

The core of the balancing loop is a PID controller that takes the current tilt angle and outputs a torque command in Nm, sent directly to the ODESC in torque control mode.

```
error = setpoint - angle  
P      = Kp * error  
I      = Ki * clamp(integral + error * dt, ±integral_limit)  
D      = -(Kd * gyro_rad_s)  
output = clamp(P + I + D, -10.0, 10.0)
```

The proportional term drives the reaction wheel in proportion to how far the bicycle is leaning. The integral term corrects for any steady-state lean that the proportional term alone cannot eliminate, for example if the bicycle's center of mass is not perfectly aligned with the setpoint. The integral accumulation is clamped to a limit of 1.0 rad of accumulated error to prevent windup during falls or extended disturbances.

The derivative term is worth examining in detail. Rather than computing the derivative of the error by differencing successive error values, the raw gyro rate is used directly and negated:



```
float d_term = -(pid->Kd * gyro_rad_s);
```

This is known as derivative on measurement. Differencing the error to get a derivative introduces a large transient spike whenever the setpoint changes, since the error jumps discontinuously. Using the gyro rate directly avoids this entirely, and it is physically meaningful: the gyro rate is the angular velocity of the bicycle, which is exactly the quantity the derivative term is trying to damp. A positive gyro rate means the bicycle is falling in one direction, and the negative sign ensures the derivative term pushes back against that motion.

The ODESC is configured in torque control mode, meaning the STM32 PID is the only control loop in the system. The ODESC's internal velocity controller is completely bypassed. This gives lower latency than velocity mode since there is no inner loop adding delay, but it also means the PID gains operate directly on torque output in Nm rather than on a velocity setpoint, which affects how the gains are interpreted and tuned.

PID Tuning

Tuning the PID gains was an iterative process that combined a structured method with hands-on observation. The process began with the integral and derivative terms set to zero, leaving only the proportional gain active. This isolates the P term and makes its effect easy to observe: too low and the bicycle falls without meaningful resistance, too high and it begins to oscillate.

Starting from a low K_p value and increasing gradually, sustained oscillation was observed at around $K_p = 850$. At this point the bicycle would oscillate once or twice and then saturate, meaning the reaction wheel would hit its torque limit and be unable to recover. This is the critical gain in the Ziegler-Nichols tuning method. Using the Ziegler-Nichols closed-loop rules, the proportional gain is set to roughly half the critical gain and the integral and derivative terms are calculated from the oscillation period. This gave a starting point for all three gains.

From there, manual refinement was used to improve the feel of the system. The Ziegler-Nichols values tend to produce an aggressive response that can be too reactive for a mechanically compliant system like a bicycle on a floor surface. K_p was brought down from the Ziegler-Nichols value to reduce the tendency to saturate at small angles. K_i was kept small to provide steady-state correction without introducing slow oscillation from integrator windup. K_d was kept low since the gyro rate already provides a degree of natural damping through the derivative on measurement approach.

Possible Improvements

The balancing system had clear limitations that point toward what would need to change to achieve more robust and sustained balance.

The most significant limitation is that angle estimation relies entirely on the IMU with no additional state information. A more robust approach would incorporate wheel encoder feedback to estimate the bicycle's lean rate independently of the IMU, which would make the controller less sensitive to IMU drift and vibration. An extended Kalman filter using both the IMU and encoder data would give a more accurate and reliable state estimate than the current gyro-weighted fusion.

The setpoint being fixed at 0.0 radians assumes the bicycle's center of mass is perfectly aligned with the geometric center, which is unlikely given the asymmetric placement of electronics and batteries. In practice the true balance point is some small nonzero angle, and the controller is always fighting a small steady-state offset that the integral term has to compensate for. Measuring and setting the correct setpoint trim would reduce the load on the integral term and improve stability.

Forward motion introduces additional dynamics that the current controller does not account for. A moving bicycle has speed-dependent gyroscopic effects from the wheels that change the effective balance point, and the coupling between steering angle and lean angle becomes significant. Extending the controller to handle forward motion would likely require a more complete dynamic model of the bicycle rather than the simplified inverted pendulum treatment used here, potentially a full state-space controller with speed as an additional input.

Finally, the reaction wheel will eventually saturate if the bicycle accumulates angular momentum in one direction over time. A desaturation strategy, where the controller periodically unloads the wheel while keeping the bicycle balanced, would be needed for any long-duration operation.

Drive, Steering, and Braking

The drive system uses a brushed DC motor connected to the rear wheel through a 25H chain and a 90T/11T sprocket reduction, giving a gear ratio of approximately 8.2:1. This

reduction prioritizes torque over top speed, appropriate for a bicycle intended to operate at low speeds under load. The motor is controlled by a 110A bidirectional brushed ESC operating at 24V, which accepts a standard PWM signal with a pulse width range of 0.8 ms to 2.2 ms and a neutral midpoint at 1.5 ms. The ESC supports motor braking through its H-bridge circuitry, meaning that when the throttle signal returns to the neutral position the motor is actively shorted rather than allowed to freewheel. This eliminates the need for a mechanical brake and keeps the braking system entirely electronic.

Steering is handled by a 160 kg-cm 24V digital servo mounted above the top tube, connected to the fork through a 1:1 3D printed spur gear arrangement. The 1:1 ratio means steering angle commands map directly to fork rotation without any scaling, keeping the relationship between commanded angle and physical response straightforward. The servo accepts a standard PWM signal in the same range as the ESC.

Both the drive and steering subsystems were tested by sending PWM commands from a Raspberry Pi, allowing manual control of speed and steering direction to verify that the mechanical and electrical systems were functioning correctly before integrating them into the broader autonomous navigation stack.

SLAM and Autonomous Navigation

The Problem SLAM Solves

For a bicycle to navigate autonomously it needs to answer two questions simultaneously: where am I, and what does the environment around me look like? These questions are deeply coupled. Building a map requires knowing your position so you can place observations correctly, but knowing your position requires a map to localize against. This circular dependency is the core challenge that SLAM addresses.

GPS is the obvious alternative for localization, but it is insufficient for the kind of precise, reactive navigation an autonomous bicycle requires. GPS accuracy in consumer hardware is typically in the range of 2 to 5 meters, which is far too coarse for a bicycle navigating around obstacles or following a specific path. It also fails entirely indoors and degrades significantly in urban environments where signal reflections off buildings corrupt the measurement. SLAM sidesteps these limitations by building and using a local map derived entirely from onboard sensors, making it self-contained and environment-agnostic.

ORB-SLAM3

ORB-SLAM3 is a feature-based visual-inertial SLAM system that works with monocular, stereo, or RGB-D cameras, optionally fused with IMU data.

Feature-based SLAM works by identifying distinctive points in an image and describing each with a compact numerical descriptor. When the camera moves, the same physical points in the scene produce similar descriptors, allowing the system to match observations across frames and infer how the camera has moved. ORB-SLAM3 uses ORB features, which are fast to compute and robust to rotation and scale changes, making them well suited to a moving platform.

The system runs three parallel threads. The tracking thread processes every incoming frame, matches features against a local map, and produces a real-time pose estimate. The local mapping thread refines the 3D map using selected keyframes and optimizes the consistency between map points and camera poses. The loop closing thread detects when the camera revisits a previously mapped area and corrects any accumulated drift in the map. The Atlas system manages multiple disconnected maps, allowing the system to recover gracefully from tracking loss rather than failing entirely.

The visual-inertial mode is the most relevant configuration for this bicycle. By tightly coupling camera observations with IMU measurements, the system gains metric scale and gravity direction that the camera alone cannot reliably provide, and inter-frame motion estimates that improve tracking robustness during vibration or rapid movement.

Sensor Configuration

The ZED 2i captures synchronized left and right images at a known baseline separation. This stereo configuration allows the system to directly triangulate the depth of observed features from a single frame, giving metric scale immediately without relying purely on motion as a monocular system would. The ZED 2i's built-in IMU provides the inertial data for visual-inertial mode, and ORB-SLAM3 would be configured in stereo-inertial mode on this bicycle using both inputs simultaneously.

The 2D LiDAR operates alongside the camera rather than contributing to the SLAM map directly. It publishes scan data to ROS at high frequency, feeding into a local costmap that sits on top of the global map produced by ORB-SLAM3. The local planner checks this costmap before executing each step of the planned path, meaning a sudden obstacle can trigger a local replanning response without invalidating the global route. There is also a latency advantage: the LiDAR can detect and react to an obstacle faster than the camera-based system since it requires no feature extraction or tracking. One limitation is that the LiDAR scans at a single height plane, so low objects or overhead

obstacles would not be detected, and the camera system would need to cover those cases.

Path Planning and Integration Architecture

Once ORB-SLAM3 is providing a real-time pose estimate and a map, a path planner computes a route from the bicycle's current position to a target waypoint. Rapidly-exploring Random Tree, or RRT, is a well suited algorithm for this application. RRT builds a tree of reachable positions by randomly sampling the free space and extending the tree toward each sample from the nearest existing node. Because the tree grows by simulating actual motion steps, it naturally respects the constraint that a bicycle can only move in the direction it is pointed and cannot turn on the spot. The random sampling can be biased toward the goal to make the search more efficient, and the resulting paths tend to be smooth enough for a steering controller to follow.

In practice this would be implemented within ROS on the Jetson Orin Nano. The ROS Navigation Stack integrates global planning, local planning, and costmap management into a single framework. The Jetson takes the pose estimate from ORB-SLAM3, computes the required steering and speed, and sends PWM signals to the ESC and servo accordingly.

The balancing subsystem sits entirely outside this data flow. The STM32 runs its closed-loop control independently at 400 Hz with no involvement from the Jetson. However there is an important coupling that any real implementation would need to address: the navigation layer cannot issue arbitrary steering and speed commands without considering the balance constraints of the bicycle. Sharp steering at low speed could destabilize the bicycle before the balancing controller can compensate. In a fully realized system the interface between the navigation layer and the drive subsystem would need to limit steering rate and curvature as a function of speed, ensuring commanded trajectories are compatible with what the balancing controller can handle.

Challenges Specific to a Bicycle

A bicycle can only move in the direction it is pointed and cannot strafe or turn on the spot, which constrains the kinds of paths the planner can generate. The path planner must account for the bicycle's minimum turning radius at a given speed, and planned paths must be smooth enough that the steering controller can follow them without inducing large lean angles.

At low speeds the gyroscopic stabilization from the wheels is negligible, placing the full burden of stability on the reaction wheel. This means the navigation system cannot treat the bicycle as a simple mobile robot: speed, steering, and balance are all coupled, and any viable autonomous navigation implementation would need to respect that coupling at every level of the stack, from path planning down to the hardware interface.

Reflection

This project was my first real exposure to embedded systems development. Coming from a background with higher-level programming environments, the shift to bare-metal firmware on a microcontroller required a different way of thinking about software. Small details that would be invisible in other contexts, such as clock configuration, SPI mode settings, or the ordering of peripheral initialization, had immediate and often hard to diagnose consequences. Early in the project I spent a significant amount of time debugging what turned out to be a single incorrect PLL value. Working through that process taught me a lot about how to approach embedded debugging systematically: reading datasheets carefully, isolating variables, and using tools like USB CDC telemetry to get visibility into what the system is actually doing rather than what you expect it to be doing.

The hardware iteration on the reaction wheel motor was a good lesson in how quickly a component selection can become a bottleneck. The original motor's torque limitation wasn't obvious until it was tested under real conditions, and recognizing that the problem was in the hardware rather than the firmware required stepping back from the code entirely.

The SLAM research was a different kind of learning experience. Working through ORB-SLAM3 independently gave me the space to understand the material properly rather than just implementing it. The visual-inertial fusion approach was something I found particularly interesting: tightly coupling two imperfect sensors through a joint optimization produces something meaningfully better than either sensor alone, and it connects directly to the same sensor fusion philosophy behind the angle estimator in the balancing controller.

Conclusion

This report has documented the hardware and firmware work undertaken during the ASB project alongside an independent study of autonomous navigation through SLAM.

The bicycle went through meaningful iteration across the reaction wheel motor, motor controller, and firmware before reaching a functional balancing state. Each hardware decision built a more concrete understanding of how a control system behaves in practice, and the debugging process at each stage deepened that understanding further.

The SLAM study examined what would be required to take the bicycle beyond balancing and into autonomous navigation. Working through ORB-SLAM3, path planning, and the integration architecture made it clear that autonomous navigation on a bicycle is a

harder problem than it is on a conventional ground vehicle, precisely because balance, steering, and speed cannot be treated independently.

The project was a broad introduction to embedded systems, control theory, and robotics, and working through it at a hardware level has opened a much larger area of engineering that I feel equipped to continue exploring.