Inverted Pendulum: Self-Balancing Robot

Ahmad Malik, EE'24
The Cooper Union for the Advancement of Science and Art
Department of Electrical Engineering
New York, USA
ahmad.malik@cooper.edu

Nicholas Singh, EE'24
The Cooper Union for the Advancement of Science and Art
Department of Electrical Engineering
New York, USA
nicholas.singh@cooper.edu

Abstract— **This project focuses on the design and construction of a self-balancing robot controlled by a PIC microcontroller. The primary objective is to engineer a robust robot capable of real-time equilibrium, demonstrating the integration of robotics, control theory, and engineering principles. The core features of the robot include a system for balancing itself upright and another for stabilizing a load-carrying tray. Moreover, it has the capability for user-controlled remote operation via a transmitter/receiver system. The key components enabling this robot to function are the sensory data from the IMU, responsive stepper motors for power delivery, an industry-standard microcontroller for enhanced processing power, and PID control theory which will accurately adjust the robots tilt to remain upright. By utilizing accelerometers and gyroscopes, the robot continuously monitors its tilt angle and rotational speed, similar to hoverboards and Segways. Beyond showcasing robotics and control theory principles, this project serves as an educational tool, providing hands-on insights into STEM concepts.**

Index Terms— PID Control Theory, Gyroscope, PIC

I. INRODUCTION

The evolution of self-balancing robots is underpinned by fundamental principles of control theory within systems engineering and mathematics. The PID controller, a prevalent control loop feedback mechanism extensively utilized in industrial control systems and various applications necessitating continuous, precise control, epitomizes this evolutionary process. Advancements in sensor technology and high-speed microcontrollers have propelled self-balancing robots beyond rudimentary mechanical models to sophisticated systems adept at maintaining equilibrium under diverse conditions. The landscape of self-balancing systems has witnessed notable progress, laying the groundwork for innovative developments, exemplified in our project. A notable example is the Segway Personal Transporter, introduced in the early 2000s. The Segway employs a sophisticated system of gyroscopic sensors and accelerometers to maintain balance, enabling users to navigate effortlessly by leaning forward or backward [1]. This technology has found applications in various industries, from personal transportation to security and logistics.

Similarly, in the field of robotics, Boston Dynamics has pioneered self-balancing robots such as the "Handle"[Fig.1]. This two-wheeled robot combines advanced control algorithms with efficient wheel-based locomotion,

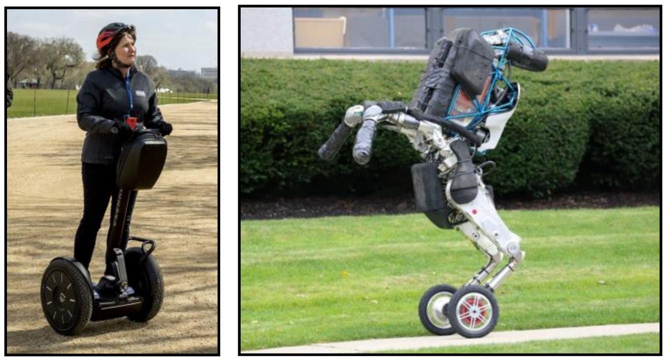


Fig. 1: Segway Personal Transporter [left] and Boston Dynamics "Handle" robot [right]

showcasing dynamic stability and agility in complex environments [2]. The incorporation of balancing technologies in such robots has expanded their potential applications in logistics, warehouse automation, and even entertainment. These existing technologies underscore the practicality and versatility of self-balancing systems. Our project builds upon this foundation, leveraging advancements in sensor technology and microcontrollers to contribute to the evolving landscape of dynamic robotic control.

Beyond academic inquiry, our project harbors potential for real-world applications. The principles underlying this self-balancing robot can be adapted for autonomous vehicles, prioritizing balance and stability. Additionally, they could serve in assistive technologies for individuals with mobility impairments, enhancing independence and quality of life. Furthermore, the project holds significant educational value, serving as a hands-on tool to impart insights across the STEM spectrum (Science, Technology, Engineering, and Math), facilitating a practical understanding of intricate concepts.

II. THE INVERTED PENDULMN PROBLEM

The inverted pendulum represents an iconic example of a dynamically unstable system. The problem consists of a mass "bob" mounted on a rigid lever that rotates about a pivot point, or fulcrum. In its upright, typical position, the "bob" remains in a state of unstable equilibrium, able to pivot and accelerate in any direction with the slightest disturbance.

The inverted pendulum's dynamic instability mirrors that of self-balancing robots, whose purpose is to remain upright in the face of a constant gravitational force and external disturbances. In order to stabilize this system, two fundamental approaches can be chosen [3]. The first option involves altering the direction of gravity, essentially changing the frame of reference. While this method might theoretically work, it is simply impractical and obviously impossible to implement in most real-world scenarios. The second and more realistic approach involves applying an external force to counteract the displacement and ensure that the restoring force is in opposition to the direction of the displacement. This method introduces controlled acceleration to the base of the inverted pendulum, in our case via the robot's wheels, allowing the "bob" to regain its vertical position.

However, it's important to note that implementing this corrective force introduces an additional challenge: as the "bob" accelerates back towards an upright position, it experiences an inertial force in the opposite direction to the wheel's movement. The magnitude of this inertial force is directly proportional to the acceleration. In this way, controlling the base of the inverted pendulum to generate the appropriate acceleration is a delicate balance of forces. It's a complicated juggling act between controlling the external force applied and managing the resulting inertial forces, all to achieve a sudo-equilibrium with the "bob" in the upright position. This control strategy forms the foundation for self-balancing robots, allowing them to navigate any challenging terrain while maintaining balance in a dynamically unstable environment. For a better understanding, Fig.2 depicts the inverted pendulum system with a mobile base [4]. L is the distance between the center of gravity of the body and the wheel axis, θ is the tilt angle of the robot, the mass of the body is m, and g is the acceleration of gravity.

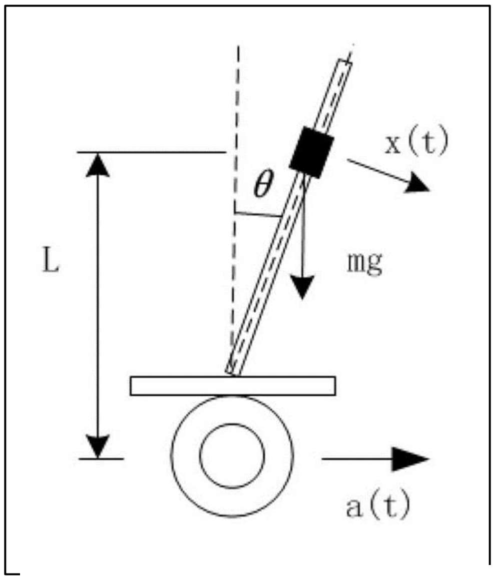


Fig. 2: Force analysis of unbalanced robot

Assuming that the angular acceleration results in $x(t)$ and the acceleration, $a(t)$, is generated by the wheels displacement, the differential Equation 1 encapsulates the forces on the "bob" in inverted pendulum system:

$$L\frac{d^2\theta(t)}{dt^2} = g\sin[\theta(t)] - a(t)\cos[\theta(t)] + Lx(t) \tag{1}$$

Considering small angles, the equation of motion can be linearized to:

$$L\frac{d^2\theta(t)}{dt^2} = g\theta(t) - a(t) + Lx(t) \tag{2}$$

When the system is in a stable equilibrium state where the wheel is not moving and the "bob" is in the vertical position, $a(t)$ is 0 so the equation simplifies to:

$$L\frac{d^2\theta(t)}{dt^2} = g\theta(t) + Lx(t) \tag{3}$$

By applying the Laplace Transform to Equation 3, we get the following transfer function.

$$H(s)=\frac{\theta(s)}{X(s)} = \frac{1}{s^2-\frac{g}{L}} \tag{4}$$

From the transfer function, the system has two poles as shown below:

$$s_p = \pm\sqrt{\frac{g}{L}} \tag{5}$$

Substituting the gravitational acceleration, 9.8 m/s^2, and assuming the center of gravity is roughly 0.27m from the center of the wheel axle, the poles of the transfer function can be plotted on the real and imaginary axis. Fig.4 shows the poles of the system on the plane, where one of the poles is located in the right half of the plane and the other pole is mirrored about the imaginary axis onto the left half of the plane. Due to the existence of a pole on the positive side of the "real" axis, the force analysis proves the system is in an unstable state. To shift the poles to negative side so that the system can be stable, we can introduce a negative feedback. One method to stabilize the system is using a proportional-integral-derivative (PID) feedback control [4].

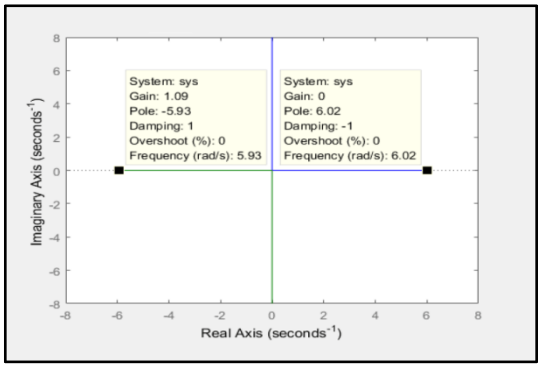


Fig. 4: MATLAB s-plane pole-plot shows unstable system

III. PID CONTROL THEORY

As mentioned in the previous section, PID is one way to stabilize an unbalanced system. PID control theory is a control mechanism used to regulate and maintain a desired state or setpoint in a system. It continuously adjusts an output based on the difference between the desired state and the current state of the system. PID is made up of three components: proportion, integral, and derivative. The proportional (Kp) component responds to the current error (the difference between the desired angle and the measured angle for our case) and generates an immediate corrective action that is linear in nature. The integral (Ki) component addresses steady-state errors by accumulating past errors to eventually overcome and correct them. The derivative (Kd) component counteracts overshoot, reducing rapid changes in the error. All of these components scale based on the current error of the system $e(t)$ and accumulate to a single control signal at any given time, $U(t)$, which can be defined as:

$$U(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt} \tag{6}$$

Since e(t) represents the error at time t, for the inverted pendulum problem the error would be the difference between the desired angle and the measured angle. Kp, Ki, and Kd are the controller gains for the proportional, integral, and derivative terms, respectively. Controller gains represent the magnitude of each component's contribution to the control output. A numerical change in any individual coefficient, Kp, Ki or Kd, changes only the size of the contribution in the path of the term. For example, if the value of Kd is changed, then only the size of the derivative action changes, and this change is decoupled and independent from the size of the proportional and integral terms [3],[5]. This decoupling of the three terms is a consequence of the parallel architecture of the PID controller as shown in Fig. [5].

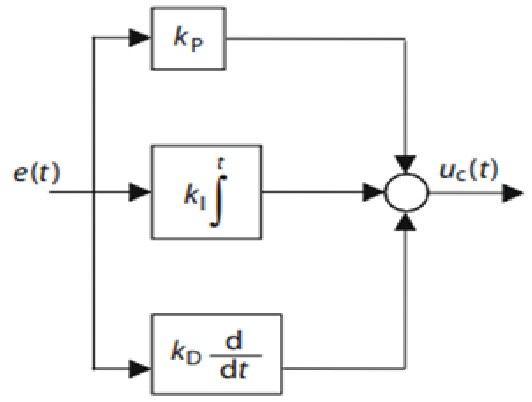


Fig. 5: Parallel, time domain formula for PID controller [5]

Building off of this, PID can be applied to the inverted pendulum problem to yield a stable solution. Previously the transfer function contained an unstable pole; this can be remedied with negative feedback [4].

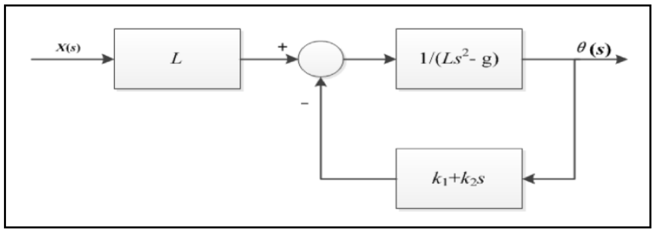


Fig. 6: Transfer function contains negative feedback (K_1+K_2 s) [4]

Here, the terms K_1 and K_2 represent the proportional and differential terms, respectively, of the PID feedback system; there is no integral term in this example so it can be assumed it is near zero and isn't necessary to stabilize the inverted pendulum system. Continuing the previous derivation, we substitute the new feedback into the transfer function:

$$H(s) = \frac{\theta(s)}{X(s)} = L\left(\frac{\left(\frac{1}{Ls^2-g}\right)}{1+\left(\frac{1}{Ls^2-g}\right)\left(\frac{K_1+K_2 s}{1}\right)}\right) = \frac{1}{s^2+\left(\frac{K_2}{L}\right)s+\left(\frac{K_1-g}{L}\right)} \tag{7}$$

The transfer function yields the two poles:

$$s_p = \frac{-K_2 \pm \sqrt{K_2 - 4L(K_1 - g)}}{2L} \tag{8}$$

Assuming $K_1 >$ g and $K_2 > 0$, the poles of the function exist in the left side of the left side of the imaginary-real plane. See Fig .7.

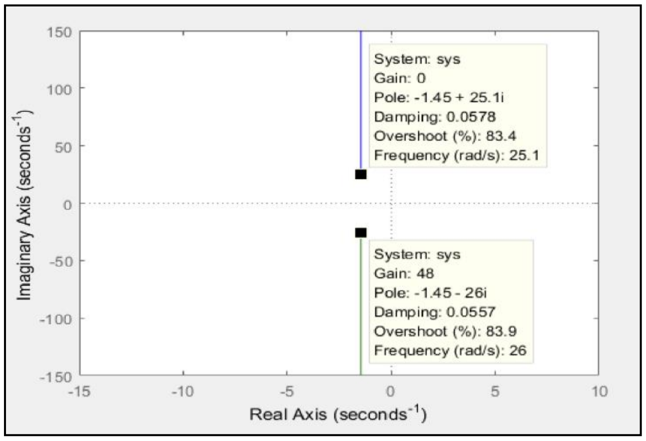


Fig. 7: MATLAB s-plane pole-plot shows a stable system [4]

From this derivation, it is apparent that introducing PID control feedback to the inverted pendulum problem will result in a stable equilibrium. The negative feedback consisting of the proportional and a derivative component of a PID control loop are enough to pull the zeroes of the transfer function to the stability region. As a side note, if a single PID component was used instead of two or three, the system would be marginally stable and the pendulum would be able to balance, although it would oscillate back and forth about the origin.

IV. RELATED WORK

Self-balancing robots remain a prominent area of research interest, showcasing a multitude of methodologies and innovations. This section offers a brief overview of notable contributions in this field, highlighting their significance alongside the distinctive features of our project.

Martins and Nunes led the project "Control System for a Self-Balancing Robot," focusing on the development of Bimbo, a self-balancing robot. Their investigation primarily aimed to assess various control algorithms' efficacy in achieving equilibrium in such robots [11]. Noteworthy was their exploration of the PID algorithms and their variable pole placement and adaptive control. The project's standout feature was its implementation of Bluetooth communication for real-time monitoring and testing, offering insight on performance.

In "Fuzzy Control of Self-Balancing Robots," Odry presented a practical learning platform for feedback control concepts, featuring a cost-effective self-balancing robot for hands-on experience [7]. Their project emphasized practical implementation of PID control feedback techniques in tuning, enhancing students' understanding of control systems. Notably, the project's emphasis on hands-on learning and fuzzy control showcased a novel approach, bridging theory and practice effectively.

While our project shares commonalities with these endeavors in self-balancing robot design and advanced control systems, it distinguishes itself through the utilization of a PIC microcontroller which is an industry standard controller with wide applications. Renowned for its robust performance and cost-effectiveness, the PIC microcontroller provides a large array of processing capabilities and communication protocols, rendering it ideal for our self-balancing robot [8]. Additionally, our project prioritizes educational value, serving as an interactive STEM learning platform, thereby setting it apart from existing initiatives.

V. ETHICAL CONSIDERATIONS

From an economic standpoint, the widespread adoption of self-balancing robots may lead to job displacement in certain industries, particularly those reliant on manual labor or manufacturing processes. As this technology matures, it becomes more affordable and accessible to organizations and businesses around the world, potentially resulting in a reduction of human labor requirements. Consequently, job displacement raises ethical questions about how to ensure fair distribution of benefits from this new technology.

The manufacturing process, including the extraction and refining of raw materials, consumes considerable amounts of energy and resources. Robots require batteries and electronics to operate, which may contribute to the growing problem of electronic waste disposal and the inhumane labor that comes with enriching precious metals. It is crucial to develop sustainable manufacturing practices that minimize societal and environmental damage.

VI. PRELIMINARY DESIGN

The objective is to build a captivating and highly stable two-wheeled balancing robot, designed for both practical use and entertainment. To achieve this, careful consideration was given to the robot's dimensions and weight. Opting for a height of approximately three feet, or "hip-level," and a width of about a foot, the robot aims for an optimal balance between stability and visual appeal. Increasing the robots height further enhances the robots stability since the poles are shifted closer to the origin as L is increased, thus resulting in a system that dampens oscillations.

During the CAD modeling design phase, we drew a connection between the robot we envisioned and "Rosey, the robot" from "The Jetsons" show. This prompted a deliberate shift in the robot's design direction to evoke a friendlier and waiter-like appearance. The robot is infused with an approachable aesthetic, suggesting companionship that not only excels in functionality but also resonates with a sense of familiarity and warmth. The CAD model in Fig.8 was designed with these notions in mind

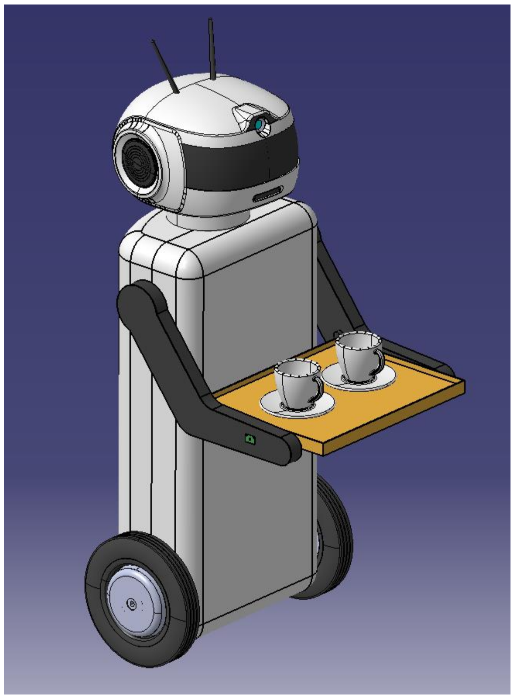


Fig. 8: CAD rendering of the self–balancing robot

The carrying capacity and weight distribution of the robot were critical considerations during the preliminary design phase. To determine the optimal parameters, we deliberated on the intended functionality, proposing a load-bearing capacity of 1 to 2 kg—sufficient for transporting beverages and meals. Given this capacity, the robot's body must possess substantial weight to counterbalance potentially uneven loads. However, an excessively heavy robot would impede maneuverability and pose a safety hazard. Conversely, a too-light structure risks instability under load displacement. Thus, a balance between power and weight was sought, resulting in a target weight of approximately 11kg without a load. With the weight and dimensions of the robot determined, the next step was to determine the electronics and hardware necessary for the robot to function well and build a frame to house it all.

Propulsion is facilitated using two high-torque stepper motors, ensuring sufficient power delivery to the wheels and load-bearing tasks. In selecting stepper motors over DC brushed motors for our project, precision and stability were of paramount importance. Stepper motors offer superior precision, minimizing oscillations commonly observed in self-balancing robots utilizing DC brushed motors. We opted for NEMA23 motors that provided an impressive 3Nm of torque [9]. Although they are slightly more expensive, Closed-loop stepper motors are used over open-loop stepper motors because they feature encoders that ensure the motor does not miss any steps and ensures the stator and coils are always aligned. This is necessary because if the shaft misaligns with the coils, it results in a sudden drop in torque which would be undesirable. Therefore, closed-loop stepper motors were selected, along with their corresponding stepper drivers. To power theses stepper motors and their drivers, we selected LiFePO4 Batteries with a capacity of 10AH and a discharge rating of 3C. These batteries, operating at 12v (24v in series, necessary for driver), deliver sufficient power/apps for approximately an hour, meeting our requirements.

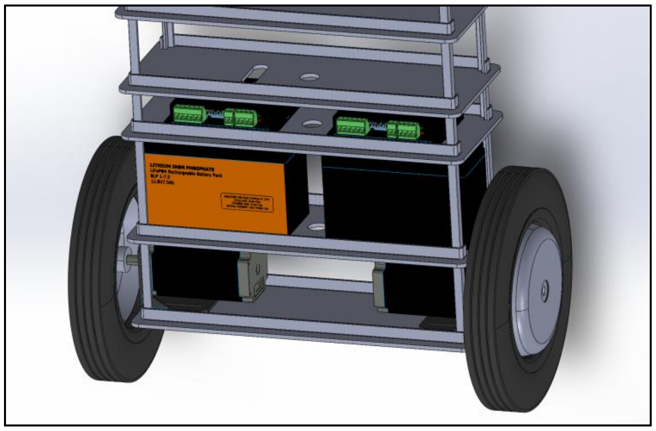


Fig. 9: CAD rendering of internal frame with driver, batteries, motors, and wheels mounted to frame.

A wheel diameter of 5 inches was chosen to optimize power transfer from the motor shaft to the ground, with a 10x1.75” Semi-Pneumatic tire selected for its traction and aesthetic appeal. For mounting the wheels securely onto the motor shafts, we employed flange couplers welded to the wheels and shaft, ensuring a stable connection.

In conceptualizing the robot's load-carrying capabilities, we initially considered a fixed basket or tray attached to the robot's body. However, inspired by "Rosey, the Robot," we adopted a design where the tray operates on its own rotating axis independently of the robot's orientation. This enhanced the robot's user-friendliness and by proactively preventing spills or falls of carried items. The arm control mechanism would involve a lever that pivots in response to the position of a threaded rod. Driven by a low-power NEMA17 stepper motor, it would supply 59Ncm of torque, sufficient for this mechanism. This design has the advantage of allow the arms to remain fixed even when the motor is not powered, lowering power demand and risk of arms dropping the tray. Fig. 10 illustrates the envisioned mechanism.

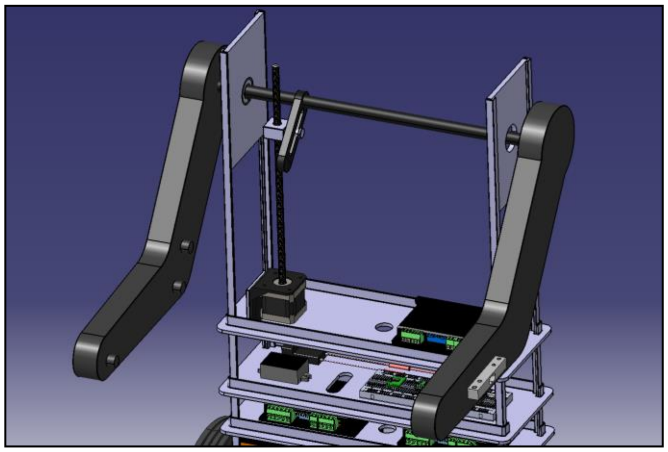


Fig. 10: a threaded rod fed from the NEMA17 Stepper Motor controls the arms /tray pivot angle.

Shifting focus to the electronics, the robot's 24V power supply necessitates conversion to 5V for sensitive electronics and monitoring sensors. To achieve this, we selected a 15W Step Down converter to regulate the voltage effectively. For precise motion sensing, the MPU6050 will serve as the inertial measurement unit (IMU), delivering a 3-axis gyroscope and accelerometer within a single chip, plus I2C support enabling fast communication with the PIC18. Two MPU6050 chips will be utilized: one for monitoring the robot's body orientation to maintain balance, and another for tracking arm positions to ensure the tray is always level.

At the heart of the robot's control system overseeing all control signals and robot functions is the PIC18F4550 microcontroller featuring 35 I/O pins, rapid analog to digital conversions, with inbuilt I2C and UART protocol support.

To establish communication with the user, the FS-iA6B receiver interfaces with any compatible 2.4GHz transmitter, outputting 6 PWM channels for comprehensive control over the robot's operations and functions. This receiver ensures robust communication within the robot's control system, enabling seamless interaction with external control devices. Figure 11 shows the electronics bay where most of these items would be mounted. Not shown are the two relays that would regulate power to arms and wheels, acting as remote controlled safety switches.

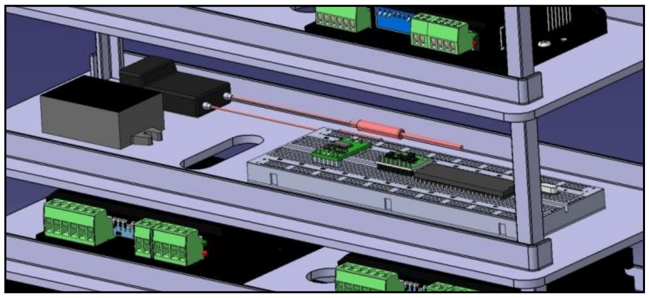


Fig. 11: Electronics bay featuring microcontroller, IMU, receiver, buck converter, and FTDI Serial to USB adapter.

VII. HARDWARE COMMUNICATION PROTOCOLS

This project involves several communication protocols implemented between various hardware components. These components include the inertial measurement unit (IMU), the PIC18 microcontroller, a 2.4 GHz receiver, a FTDI Serial to USB adapter, and a PICKIT2 chip programmer. Almost all of the communication revolves around the microcontroller, serving as the central hub for all data transmission and reception. The microcontroller communicates with the IMU to receive sensory data which it processes through control algorithms and transmits pulses to the stepper motor drivers. These drivers, in turn, power the coils or phases of their corresponding stepper motors, resulting in motor/wheel rotation. The robot also features remote control capability, receiving signals from the 2.4GHz receiver which feed into the control algorithms that output the desired motion.

During the testing phases of this robot, the microcontroller would transmit data through the FTDI Serial to USB adapter which helped to better visualize the data it was receiving from the IMU.

Each hardware component requires a communication protocol governing their interaction between the microcontroller and peripheral devices. These communication protocols are: I2C, SPI, USB, PWM, UART, and STEP/DIR. Figure 12 show the various components and their respective communication protocols. The inter-integrated circuit (I2C) protocol facilitates the exchange of data between the microcontroller and the IMU. Pulse width modulation

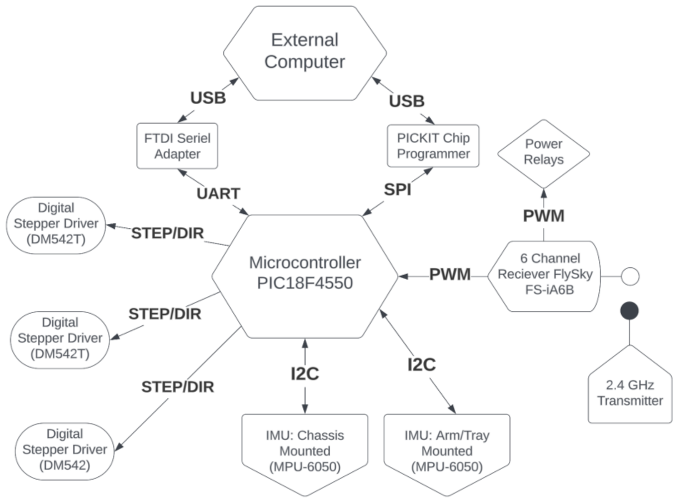

External Computer
USB
USB
FTDI Seriel Adapter
PICKIT Chip Programmer
Power Relays
UART
SPI
PWM
Digital Stepper Driver (DM542T)
STEP/DIR
Microcontroller PIC18F4550
PWM
6 Channel Reciever FlySky FS-iA6B
STEP/DIR
Digital Stepper Driver (DM542T)
I2C
I2C
STEP/DIR
2.4 GHz Transmitter
Digital Stepper Driver (DM542)
IMU: Chassis Mounted (MPU-6050)
IMU: Arm/Tray Mounted (MPU-6050)

Fig. 12: Hardware components and their respective communication protocols

(PWM) is used to regulate the power supplied to the stepper motor by disabling and enabling relays. In addition, PWM protocol is used to communicate directional control to the microcontroller from the separate channels of the 2.4GHz transmitter/receiver. The Step/Direction (STEP/DIR) protocol dictates the movement of the stepper motor by conveying step pulses and directional signals to their respective drivers. Universal asynchronous receiver/transmitter (UART) protocol is used to serially send data to the FTDI Serial to USB adapter, whose data can be read using an external computer through USB interface. Finally, serial peripheral interface (SPI) is utilized for high-speed, full-duplex communication between the microcontroller and PICKIT chip programmer which flashed the memory of the PIC18 whenever serial data is transferred from an external computer through USB. Other than SPI and USB which come pre-programmed in their respective devices (PICKIT programmer and FTDI adapter), all other protocols have been directly programmed into the microcontroller to ensure successful communication between hardware devices.

VIII. FABRICATION

The fabrication process was completed in three phases: chassis and body, electronics integration, arm and tray mechanism. During each phase, all mechanical and electronic components were thoroughly tested for functionality before moving to the next step.

A. *Chassis and Body*

The chassis is built around a 13.5” x 5.5” block of laminated wood with a 5/8” thickness to reduce as much flex and strain wear that will come from the motors. On the ends of the farther ends of the wooden block, the motor mounting brackets are mounted for the NEMA23 stepper motors. Once carefully centered and positioned, six dry wall screws are used to secure each mounting plate to the wooden block. Each stepper motor is screwed in with four M4 bolts to the mounting bracket. This forms the load-bearing portion of the robots chassis as shown in Fig. 13. The 10mm to 35mm flange couples are welded to the hubs of the wheels. They are then welded to the motor shafts to ensure an extremely strong connection between motor and wheel. Fig. 14 shows the wheels mounted on the motor shafts.

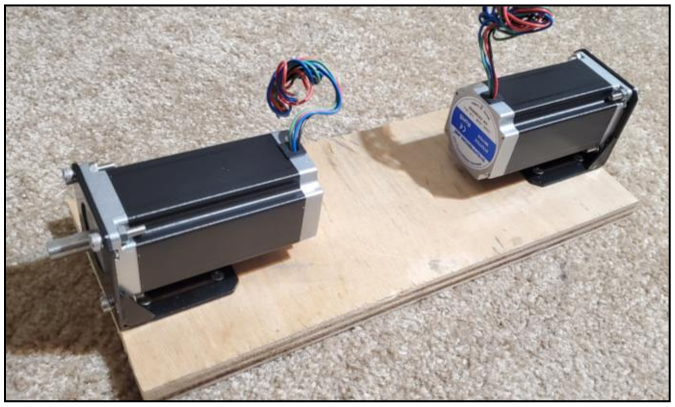


Fig. 13: Stepper motors mounted to brackets and wooden base

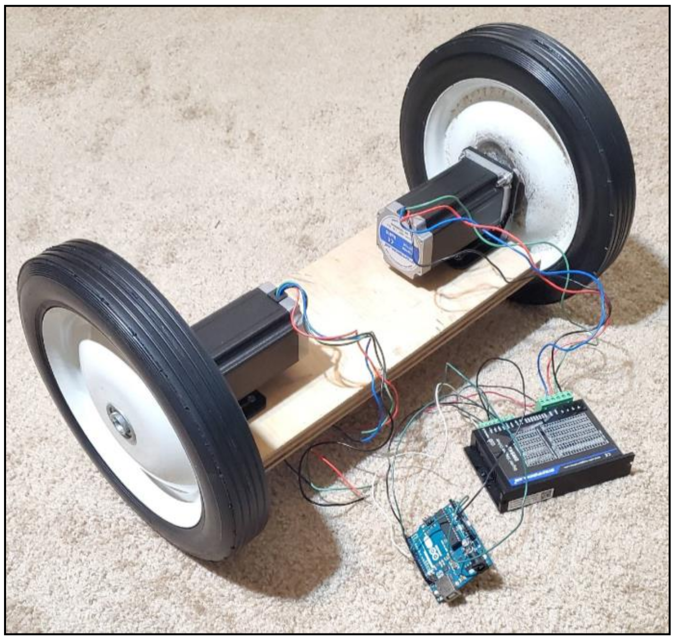
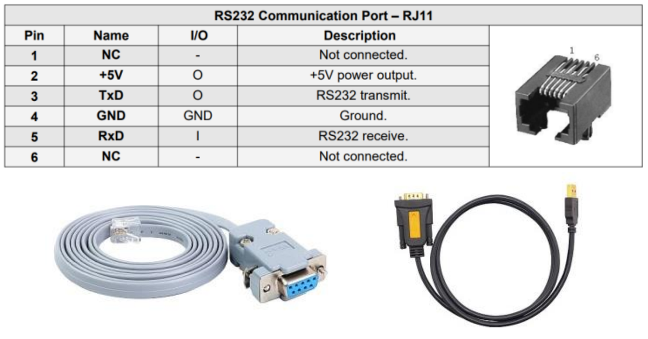


Fig. 14: Wheels welded to motor shafts and undergoing testing

Testing with an Arduino Uno and stepper driver reveals that the drivers have a spring-like, oscillatory behavior when the motor shafts loaded with the inertia of the heavy wheels and given an rotational error to correct. This problem can be mediated by tuning the motor drivers through the serial RS232 port on the side of the drivers. Since the drivers have their own PID control loop to correct for error, the PID constants can be modified to better suite their application using the software that comes with the stepper drivers. By default, the drivers a preloaded with PID values that correspond with typical lathe or milling operations (high torque, but low inertia). The cable that connects to these stepper drivers to a computer running the software requires a male RJ11 connector to RS232 9-pin connector which is then fitted to adapter that converts it into a USB adapter running on the PL2303 prolific chipset. See Fig 15 for the required cable adapters and pinot of the stepper driver RJ11 serial communication port.


RS232 Communication Port – RJ11

Pin	Name	I/O	Description
1	NC	-	Not connected.
2	+5V	O	+5V power output.
3	TxD	O	RS232 transmit.
4	GND	GND	Ground.
5	RxD	I	RS232 receive.
6	NC	-	Not connected.

Fig. 15: "Top" is the RJ11 port pinout of stepper driver serial port. from datasheet [10]. RJ11 to RS232 adapter is "bottom left" and RS232 to USB serial adapter on "bottom right".

At the time of fabrication, the steps involving tuning the motors was overlooked because the necessary cables and adapters were not in-hand and it was assumed the tuning could be done later if was even necessary. This later reveals to be a major setback as the robot has tendency to uncontrollable oscillate when given large inputs due to driver miss-tuning.

Continuing the fabrication process, the rest of the body is built upon the chassis. This involves four 15mm aluminum extrusions to be cut to equally lengths of 375mm and mounted with L-brackets that are bolted and screwed into the wooden base and extrusion. Four shelves were manufactured using a 1/8" thick aluminum composite sheet along with a pair of 1/4" aluminum angles on each shelf which provided added support. These shelf components were riveted together using 3/16" pop rivets. Once fabricated, the shelves were roughly equally spaced and fastened to the vertical extrusions. See Fig. 16.

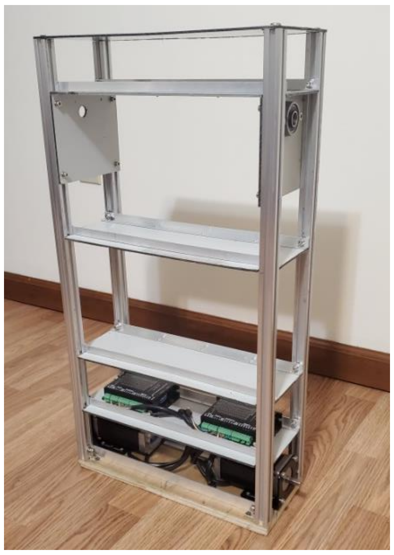


Fig. 16: robot body and shelves

Figure 16 also shows the 15mm ball bearings that were fastened in place to each side of the robot using the same aluminum composite material used for the shelves. This will be used to hold the rod that will rotate the arms. A 15mm aluminum rod is passed through the body and ball bearings. A 1:60 teeth worm-gear is slid onto the rod and mounted in place using set screws. The "worm" of the worm-gear mechanism is tightened with set screws to the NEMA17 stepper motor which is mounted to its corresponding motor mount bracket using M3 screws. The motor and bracket assembly is mounted vertically to the side using M3 nuts and bolts. See figure 17 for this completed assembly.

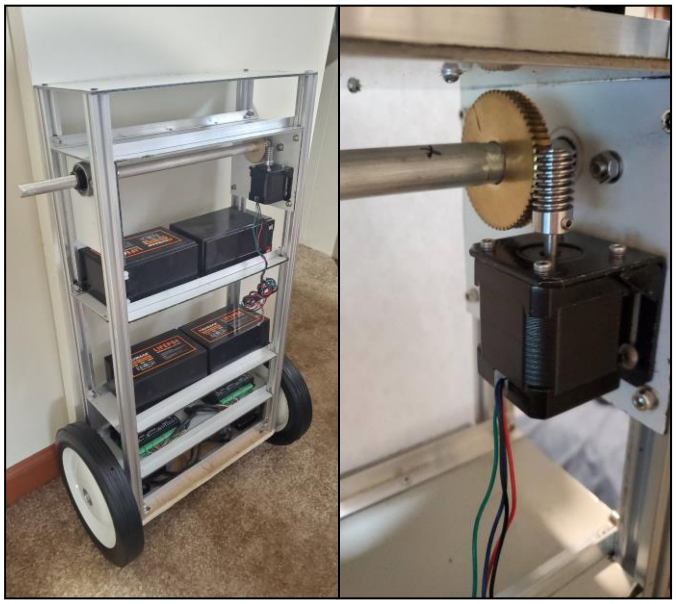


Fig. 17: Arm rod and worm-gear mechanism

B. *Electronics Integration*

As seen in previous figures, the motor drives are screwed into place on second shelf leaving the upper three shelves vacant. To maximize stability, two of the upper shelves will be dedicated towards holding the batteries due to size weight. It is not intuitive, but forcing the center of gravity in an inverted pendulum system increases the overall stability of the system. This is because it increases the inertia of the upper half, giving the lower half (wheels) more time to correct and adjust the systems tilt. This will also help us leverage and counter act any weight that may be placed on the arms. Once batteries are glued in place, the lower, middle shelf is utilized to house most of the soft electronic systems. This includes the relays, RC receiver, 5v buck converter, breadboard, PIC18, FTDI board, PICKIT programmer. The MPU6050 is not mounted in this area; rather it is moved as high as possible (fourth shelf) to receive the most accurate sensory data. A shielded wire is used to connect the MPU6050 to the PIC18 to minimize noise coming from the stepper motors. This was discovered during testing phases in which the MPU6050 would reset itself if enough motor noise was present and solid wiring was used instead of shielding. Included in the electronics shelf is the driver for the arm stepper motor, screwed in place in the upper corner. See figure 18 for the completed electronics bay.

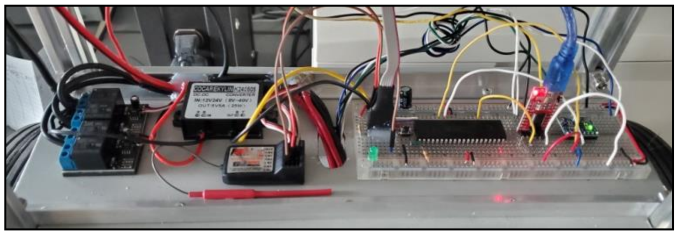


Fig. 18: electronics bay on second shelf

As an additional safety feature, an LCD display is used to show the system voltage and current/ power draw, and is mounted on the third shelf. Next to it is the main power switch that controls power to whole robot. Figure 19 shows the completed chassis, body, and all electronics that have been wired, with the exception of the arm IMU which is mounted in the next phase.

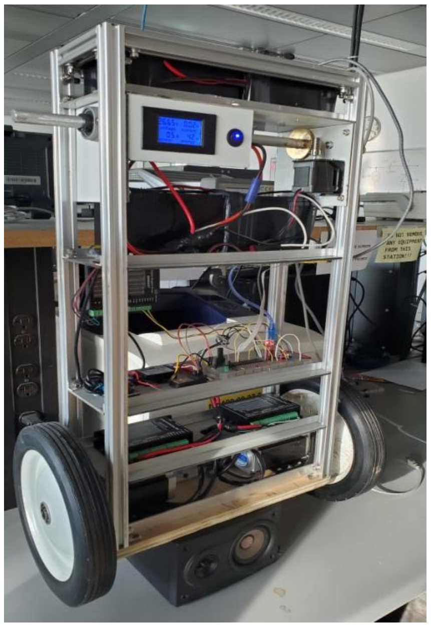


Fig. 19. Completed chassis, body, and almost all electronics are wired

C. *Arm/ Tray Mechanism*

At this point the robot has everything it needs to balance mechanically and electronically. To complete our design, this robot has arms and a tray to independently balance whatever is placed on the tray. The first step is to use the robots CAD design to print a template of the arm design. Using this template, two arms were cut from 5/8" thick wood and a hole was drilled where the 15mm arm rod would go through snuggly on each side of the robot. Once attached with friction, a single long screw would be used to firmly connect wooden arm with the rod. Epoxy was added for added strength. The other arm was connected the same way and eyeballed to match the leveling of the other arm.

Once the arms are secured, a 8” x 13” tray is fabricated using the 1/8” aluminum composite, and 1/4” aluminum angles are riveted to the trays sides for added stiffness. This tray is then fastened to the arms using two long M3 bolts on each arm. A second MPU6050 sensor is wired and glued under the arms. Finally, a 3D printed PLA head is added to the robot completing the robot fabrication (Fig. 20).

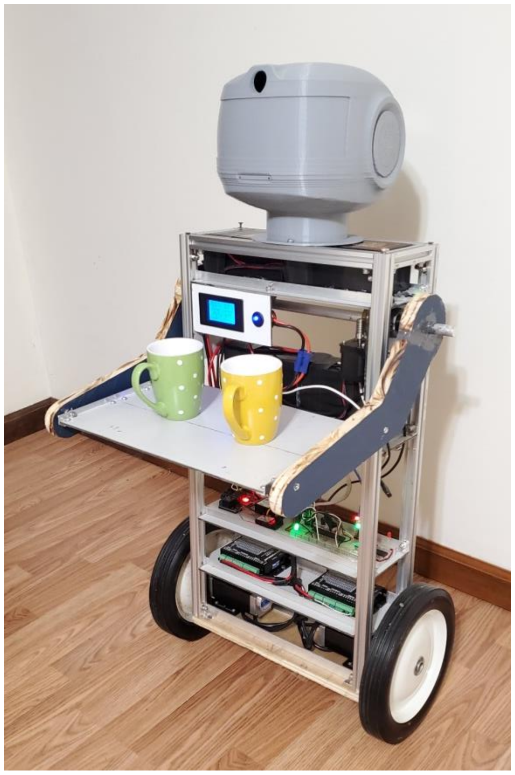
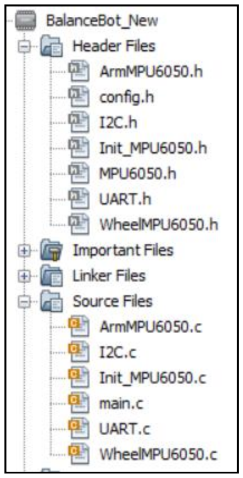


Fig. 20: robot fully fabricated

IX. CODE IMPLEMENTATION

Before writing code to the PIC18, it must be configured correctly so that the microcontroller can startup and utilize the correct settings. By using MPLABS built-in configuration bits generator, we are able to set the correct settings such as disabling the watchdog timer, setting PORTD as I/O, setting the clock frequency to 8MHz, etc. These definitions are generated and saved in the config.h header file as shown in Figure 21.



```
BalanceBot_New
  Header Files
    ArmMPU6050.h
    config.h
    I2C.h
    Init_MPU6050.h
    MPU6050.h
    UART.h
    WheelMPU6050.h
  Important Files
  Linker Files
  Source Files
    ArmMPU6050.c
    I2C.c
    Init_MPU6050.c
    main.c
    UART.c
    WheelMPU6050.c
```

Fig. 21: Project code tree

A. *Angle Measurements*

The first task is to write the necessary code for speaking with the MPU6050. To receive angle measurements or to configure the IMU, the PIC18 needs to communicate with it using I2C with pins 34 and 33 (SDA and SCK). An I2C library is needed to implement this which was provided by Khaled Magdy on his forum *deepbluembedded* [11]. By modifying the pin assignments for PIC18 (not PIC16 which is what Magdy was using) and increasing the baudrate to 400KHz, we have all of the necessary functions to transmit data through I2C. To start a communication on I2C, the serial data (SDA) line is pulled from a high to a low state, while the serial clock (SCL) line shifts from high to low, signaling the initiation of communication. Then the PIC18 sends out 7-bit address specifying the slave devices address, followed by a bit signifying read or write [12]. The default slave address of any MPU6050 is 0x68. Since we want two MPU6050 on the I2C bus (arms and body), we need to change the address of one of the IMUs. An easy work around accourding the MPU6050 data sheet is to wire the IMU so that +5v is going to the AD0 pin instead of the VCC pin; this automatically sets the default address of the MPU6050 as 0x69 [13]. Figure 23 shows the wiring of the two MPU6050s.

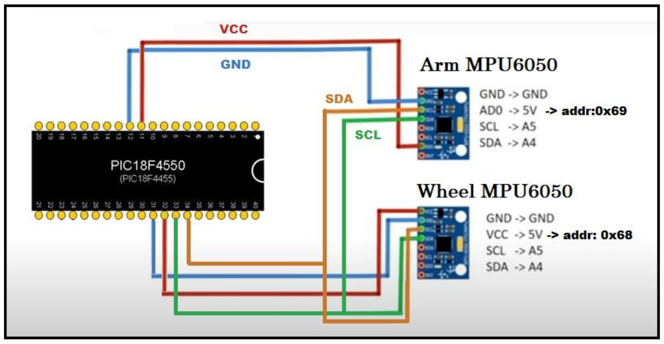
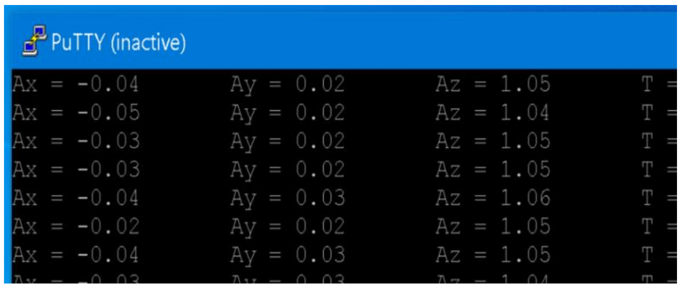


Fig. 23: Wiring of two MPU6050s to the PIC18

Using the functions in the I2C library, we are able to write addresses and data to each MPU6050 and receive data without collisions. To view this data coming out, we write to a buffer variable that holds an array of characters which is then seriel transmitted to the TX pins of the PIC18 using a UART library that was also provided by Magdy from *DeepBlueMinded.* We modified the code for high baudrate and specific registers and pins for our microcontroller. The data is received through the RX pin of the FTDI board which can then plug into a computer via USB and display the serial data through a PuTTY terminal (see Fig. 24).



```
PuTTY (inactive)
Ax = -0.04       Ay = 0.02       Az = 1.05       T =
Ax = -0.05       Ay = 0.02       Az = 1.04       T =
Ax = -0.03       Ay = 0.02       Az = 1.05       T =
Ax = -0.03       Ay = 0.02       Az = 1.05       T =
Ax = -0.04       Ay = 0.03       Az = 1.06       T =
Ax = -0.02       Ay = 0.02       Az = 1.05       T =
Ax = -0.04       Ay = 0.03       Az = 1.05       T =
Ax = -0.03       Ay = 0.03       Az = 1.04       T =
```

Fig. 24: PuTTY terminal snippet

Figure 24 shows the raw data received from the MPU6050 comprising of the accelerations, temperature, and gyro reading. The readings coming out of the FIFO register of the sensor can be modified in the "Init_MPU6050.h" file so that we only receive the desired measurements we need.

To get the angle measurements, we can either integrate the gyro twice, or use trigonometry on the acceleration. As show in figure 25, the gyro integration (in blue) results in an angle measurement that is stable but drifts over time, while the accelerometer (in red) results in a noisy measurement but does not drift like the gyro.

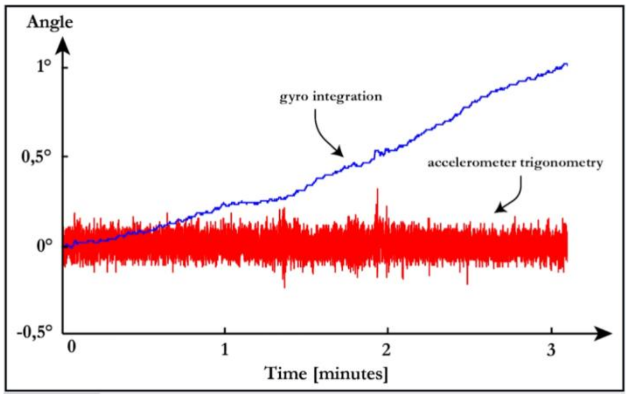


Fig. 25: gyroscope and accelerometer angle measurement [14]

To remedy these defects, a kalman filter can be implemented in our code. This filter leverages probability to take a noisy signal and filters it using an input that governs how noisy the signal should be. In this case, we will use the accelerometer angle measurement as the signal and one of the gyroscope measurements as the input to stabilize the measurement. Figure 26 shows the resulting output of the kalman filter in blue, demonstrating a dramatic improvement in the measured angle.

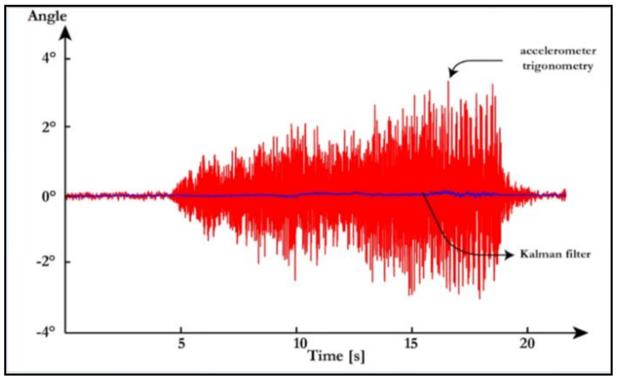
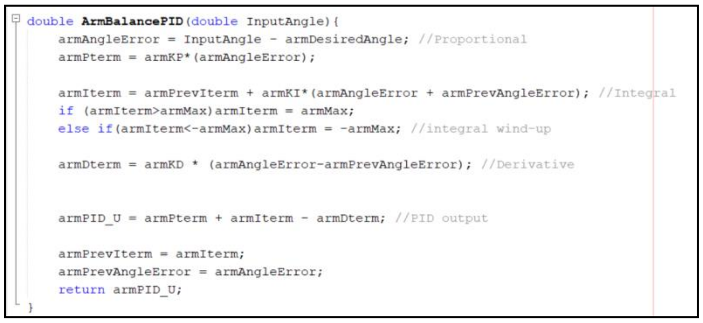


Fig. 26: kalman filter results in more accurate measurement [14]

B. *PID Code Implementation*

Now that we can accurately measure the arm and body angle, we can now write functions that can take the current measured angles compare it to the desired angles, and devise an output control signal to control the motors. The PID implementation for the wheels is shown in figure 27.



```
double ArmBalancePID(double InputAngle){
    armAngleError = InputAngle - armDesiredAngle; //Proportional
    armPterm = armKP*(armAngleError);

    armIterm = armPrevIterm + armKI*(armAngleError + armPrevAngleError); //Integral
    if (armIterm>armMax)armIterm = armMax;
    else if(armIterm<-armMax)armIterm = -armMax; //integral wind-up

    armDterm = armKD * (armAngleError-armPrevAngleError); //Derivative


    armPID_U = armPterm + armIterm - armDterm; //PID output

    armPrevIterm = armIterm;
    armPrevAngleError = armAngleError;
    return armPID_U;
}
```

Fig. 27: PID function to balance the arms

From the code, the input to the PID function is the current measured angle. Global variables are used to carry the previous angle value and the previous integral value so that the total error can be calculated. The PID function to control the wheels is almost exactly the same except for the constants governing proportional, integral, and derivative terms.

The output of the PID function is a double and can be negative or positive. In the main loop, the value is normalized and floored and sent into their respective stepper motor drive functions. The stepper motor drive functions receive a bool as an argument, which represents the direction governed by the polarity of the PID output. The other argument is the absolute value of the PID function which represents the number of steps the stepper motor will need to take. The more steps passed into the stepper motor functions, the longer each function will take to step the motors.

IX. RESULTS AND PROBLEMS

The fabrication of the robot went flawlessly and resulted in an extremely strong and robust robot that we could comfortably execute code on. In the entire span of its rigorous test, nothing failed mechanically, despite the high torque of the motors and weight of the robot. The robot could be remotely powered and disable using the transmitter and we were successful in balancing the arm/tray.

Unfortunately, the robot struggled to balance. If we limited the PID to output a low number of steps, the robot would balance well enough, but fall over if it received any disturbance due to the limited PID output. If the PID value was uncapped, the robot could balance for some time, but randomly oscillate uncontrollably. This came down two factors: slow code execution and unturned drivers. Addressing the first problem, the robot was executing instructions too slowly for the stepper motors to run well. The main loop starts by getting an angle measurement of the body, which takes 1.5ms, which it then sends to the wheel PID function, taking roughly 0.5ms. It then sends the PID value to the stepper motor function controlling the wheels.

This function can take as little as a few micro seconds to 1ms depending on the PID value (higher PID = more steps for motor). It then repeats this for the arm. Thus, between one cycle of the main loop, the wheel stepper motors are delayed by 4-5ms before new instructions are given regarding their rotational speed and displacement. This lag isn't much and can be minimized if the motors are reactive enough. However this is where the larger problem begins.

The stepper drivers, as mentioned before, are tuned to work on lathes and milling machines. This means the driver expects the motor shaft to have low inertia but deliver ample torque. Our robot needs torque but has many magnitudes higher inertia situated on the shaft, especially with the full weight of the robot. This results in a stepper driver response that often oscillates if the robots motors are requested to move rapidly; the drivers over shoot the error and then over correct. This problem would be solved if the "kd" term of the PID loops within the drivers could be increased or decreased roughly a hundred times. "kd" is the derivative term of the PID loop and it deals largely with the inertia of the system. During the development of the robot, various cables and adapters were purchased with hope that it would establish communication with driver software through USB. All attempts were unsuccessful. After careful testing, it was determined that the drivers were partially faulty since the RX line of the RS232 port was at -13v, highly irregular.

IX. CONCLUSION

Our project originally aimed to develop a self-balancing robot controlled by a PID system using a PIC18 microprocessor. Despite our efforts to integrate various components such as the Inertial Measurement Unit (IMU), I2C communication, and radio control, we faced significant challenges that prevented us from reaching stable self-balancing behavior. The primary obstacle we encountered was the difficulty in fine-tuning the PID parameters for the motor drivers. This limitation led to oscillations, particularly when the robot experienced rapid disturbances or quick instructions. Despite attempting various PID control methods, we were unable to overcome this critical roadblock. Additionally, the performance of the robot's code posed challenges, with the main loop execution time impacting the responsiveness of the control system. This issue, combined with compatibility problems with the motor drivers, contributed to the overall failure to achieve stable self-balancing. While our project did not meet its primary goal, the experience gained was exceptional. We developed hands-on skills in robotics, microcontroller programming, and troubleshooting electronic components. Moving forward, we recognize the importance of selecting compatible components and verifying their functionality before taking the next step.

References

[1] Paul Holmes, "Introducing the Segway Human Transporter," *Provoke Media,* Apr. 02, *2002*

[2] R. Team, "Handle," *ROBOTS: Your Guide to the World of Robotics*, Apr. 21, 2023

[3] Junoh, Suhardi Azliy Bin. "Two-wheeled balancing robot controller designed using PID", pp. 23-24, Jan.1, 2015

[4] Johnson, Michael A., and Mohammad H. Moradi." PID control" London, UK*: Springer-Verlag London Limited*, 2005.

[5] Zhao, J.; Li, J.; Zhou, J. "Research on Two-Round Self-Balancing Robot SLAM Based on the Gmapping Algorithm". *Sensors* 2023

[6] Martins, R. S., & Nunes, F. "Control system for a self-balancing robot". *4th Experiment@International Conference* , 2017

[7] Odry, Á., Fullér, R., Rudas, I. J., & Odry, P. "Fuzzy control of self-balancing robots: A control laboratory project", *Computer Applications in Engineering Education.* , 2020

[8] I. Bangalore and I. Bangalore, "IIES - Top Embedded Institute in Bangalore," *IIES - Indian Institute of Embedded Systems, 2018*

[9] Ningbo Smooth Electric Manufacture Co., Ltd., "How to choose the size of a stepper motor," *Ningbo Smooth Electric Manufacture Co., Ltd.*, Jan. 11, 2024.

[10] Revision 3.0 "User Manual CL57T(V3.0) Closed Loop Stepper Driver." *STEPPERONLINE,* 2017

[11] K. Magdy, "MPU6050 with Microchip PIC | Accelerometer gyroscope interfacing with PIC," *DeepBlue,* Aug. 17, 2023.

[12] "How I2C Works and I2C Bus Protocol Explained with Examples." *The Geek Pub,* 2019

[13] MPU-6000 and MPU-6050 Register Map and Descriptions Revision: *https://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/RM-MPU-6000A.pdf*

[14] "quadcopter build and programming manual", *Carbon aeronautics*, August 2022