Course Project Report

Self-Balancing Robot

Hasan Farhood, Sepehr Ahmadian, Reza Shahriari, Mohammad-Hossein Agha-Azizi, Sheida Niaparast

Abstract: In this project we have controlled a self-balancing robot using Arduino UNO, L298N, 2x DC Motors, 3x 18650 Batteries, 2x two-state switch, MPU6050. In this project we have made use of Solid Works 2023 to design our robot, we have implemented a MATLAB, Simulink to tune K_p and K_d and K_I. We have an overall iterative solution to tuning the PID parameters

I) Chapter (Phase) I

This Chapter mainly deals with simulating our design in 3D solid works 2023. We plan our design and sketch in this software based on which we built our finalized robot.

1. Introduction

How do we control Self-Balancing Robot?
Prerequisites and Components
About Solid Works:
About Our Sketch :
How did we build our sketch :
Sensor Networking : Different Topology
About Actuator :
The 6-Volt Motor
About the IMU :
What does the IMU do ?
How does it gets roll, pitch and yaw?
How does it handle noise
About the interface :
How to Watch and Monitor
How to Control

II) Chapter (Phase II) : Control

This Chapter Mainly discusses the idea behind PID control and how it works.

2. PID Control :

Experimenting with PID control
The effect of each term in the Control loop
Our main Code

Chapter (Phase) III : Simulink

In this chapter we simulate our model with pictures and illustrations

How do we control Self-Balancing Robot?

Controlling a Self-Balancing Robot: A Technological Triumph

Self-balancing robots, also known as inverted pendulum robots, are a marvel of modern robotics, relying on sophisticated control systems to maintain stability and navigate their environment. The fundamental challenge lies in ensuring that the robot can continuously adjust its position to remain upright, akin to a tightrope walker maintaining equilibrium on a thin rope. Here's a comprehensive exploration of how these robots are controlled.

1. Sensor Integration: Self-balancing robots rely heavily on sensors to gather real-time data about their orientation and motion. Inertial Measurement Units (IMUs), typically containing accelerometers and gyroscopes, play a crucial role in providing precise information about the robot's tilt and angular velocity.

2. PID Control: Proportional-Integral-Derivative (PID) control is a cornerstone in self-balancing robot algorithms. This control system continuously calculates an error signal based on the difference between the desired and actual tilt angles. The proportional component addresses the current error, the integral component deals with accumulated past errors, and the derivative component anticipates future errors. Tuning these parameters is crucial for achieving a responsive and stable control system.

3. Actuators and Motors: To physically adjust its position, a self-balancing robot employs actuators, often in the form of electric motors. These motors drive the wheels or other locomotion mechanisms. By dynamically adjusting the speed and direction of these motors based on the PID-controlled error signal, the robot corrects its balance and moves in the desired direction.

4. Microcontroller or Processor: The brain of the self-balancing robot is a microcontroller or processor that executes the control algorithms. Popular choices include Arduino boards or Raspberry Pi, which can handle the computational demands of real-time control and sensor fusion.

5. Feedback Loop: The control system operates in a continuous feedback loop. As sensors detect deviations from the upright position, the microcontroller processes this information, calculates the necessary adjustments, and commands the motors to act accordingly. This rapid feedback loop ensures that the robot can make instantaneous corrections to maintain balance.

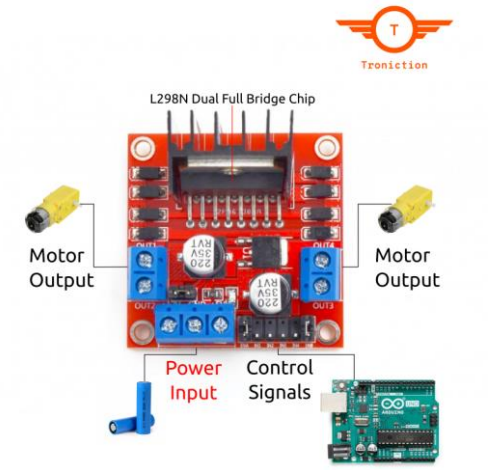
6. Learning Algorithms: Advanced self-balancing robots may incorporate machine learning algorithms, allowing them to adapt and improve their balancing performance over time. These algorithms can optimize control parameters based on experience and environmental conditions.

In conclusion, the control of a self-balancing robot is a multifaceted process that involves intricate sensor integration, PID control algorithms, precise actuators, and intelligent processors. This amalgamation of technologies enables these robots to exhibit a remarkable ability to maintain stability in diverse and dynamic environments, making them a fascinating and practical application of modern robotics.

Prerequisites and Components:

1. **Arduino UNO:**
    - The Arduino UNO is a popular microcontroller board based on the ATmega328P chip. It serves as the brain of your project, executing the programmed code to control and coordinate the various components. It has digital and analog input/output pins, making it versatile for interfacing with sensors, actuators, and other electronic components.

2. **L298N Motor Driver:**
    - The L298N is a dual H-bridge motor driver IC. It allows you to control the direction and speed of two DC motors independently. It's commonly used in projects involving motorized devices such as robots and vehicles. The L298N interprets signals from the Arduino to drive the DC motors in the desired manner.



3. **2x DC Motors:**

- DC motors are electric motors that run on direct current. In your project, these motors are likely responsible for providing movement to your robot or another device. The L298N motor driver facilitates the control of these motors by adjusting their speed and direction based on signals from the Arduino.

4. **3x 18650 Batteries:**

- 18650 batteries are lithium-ion rechargeable batteries. They are commonly used in electronic projects due to their high energy density. In your project, these batteries likely provide the power source for the DC motors and other components. Ensure that the voltage and current ratings match the requirements of your motors and electronics.

5. **2x Two-State Switch:**
    - Two-state switches, also known as toggle switches, have two possible states: on and off. These switches are likely incorporated into your project to control specific functions or to act as power switches for turning components on or off manually.

6. **MPU6050 Gyroscope and Accelerometer:**

- The MPU6050 is a sensor module containing a gyroscope and an accelerometer. It measures the rate of rotation and acceleration of your device in three dimensions. In a self-balancing robot project, the MPU6050 is often used to gather real-time data about the robot's tilt and acceleration. This data is crucial for implementing a control system (like PID) to adjust the motors and maintain balance.

By combining these components, it appears that your project involves a self-balancing robot or a similar robotic device. The Arduino UNO serves as the central controller, the L298N drives the DC motors, the 18650 batteries provide power, the two-state switches offer manual control, and the MPU6050 contributes to the robot's stability by providing feedback on its orientation and motion.

SolidWorks: Empowering 3D Design and Innovation

SolidWorks stands as a pinnacle in the realm of Computer-Aided Design (CAD), offering a robust platform that empowers engineers, architects, and designers to bring their visions to life in the digital space. Developed by Dassault Systèmes, SolidWorks has become synonymous with efficiency, precision, and user-friendly 3D modeling.

At its core, SolidWorks operates on the principle of parametric design, a revolutionary approach that allows users to define parameters and relationships between various components of a model. This parametric nature facilitates unparalleled flexibility and ease in design iterations. Changes made in one part automatically cascade throughout the entire model, providing a dynamic and responsive environment for crafting intricate designs.

One of SolidWorks' standout features is its extensive and engaged user community. The platform boasts a vast network of users who share insights, tutorials, and collaborative problem-solving through forums and user groups. This communal knowledge-sharing not only accelerates the

learning curve for newcomers but also fosters a collaborative spirit that transcends geographical boundaries.

SolidWorks is revered for its integration capabilities, particularly in the realms of simulation and manufacturing. Engineers can subject their designs to thorough analysis using integrated simulation tools, covering structural integrity, thermal dynamics, and fluid flow. Furthermore, the seamless integration with Computer-Aided Manufacturing (CAM) software ensures a smooth transition from design conceptualization to the creation of toolpaths for CNC machines.

A stronghold of SolidWorks lies in its sheet metal design capabilities, offering specialized tools for creating intricate sheet metal components. The software facilitates the creation of flat patterns, bends, and other sheet metal features with precision, catering to industries where sheet metal plays a pivotal role.

Regular updates from Dassault Systèmes ensure that SolidWorks remains at the forefront of technological advancements. Users benefit from a continuous stream of new features, bug fixes, and enhancements, reinforcing the software's position as an industry leader.

In essence, SolidWorks is not just a CAD tool; it's an enabler of innovation, a facilitator of collaboration, and a catalyst for turning imaginative concepts into tangible designs. Its impact resonates across diverse industries, shaping the landscape of 3D design and engineering with its intuitive interface and powerful capabilities.

Solid Works Main Feature:
SolidWorks boasts a myriad of features that contribute to its widespread adoption and success across industries. The software's versatility extends beyond basic 3D modeling, encompassing a range of functionalities that cater to the complex needs of modern design and engineering:

1. Advanced Assembly Modeling: SolidWorks excels in handling intricate assembly structures.

With features like top-down design and the ability to manage large assemblies efficiently, it becomes an invaluable tool for projects ranging from consumer products to industrial machinery.

2. Dynamic Motion Analysis: Engineers can simulate and analyze the motion of their designs, en-

abling them to evaluate mechanisms, test kinematics, and ensure that components move as intended. This feature aids in detecting interferences and optimizing motion for better performance.

3. Photo-Realistic Rendering: SolidWorks Visualize, an integrated rendering tool, allows users to create stunning, photo-realistic images and animations of their designs. This capability aids in visualizing the final product, presenting concepts to stakeholders, and creating marketing materials.

4. Electrical and PCB Design: SolidWorks extends its reach beyond mechanical design with integrated solutions for electrical and printed circuit board (PCB) design. This seamless integration ensures that electrical components and systems are seamlessly incorporated into the overall design process.

5. PDM (Product Data Management): SolidWorks PDM provides a centralized platform for managing and collaborating on design data. It ensures version control, facilitates collaboration among team members, and enhances overall data organization, leading to increased productivity and reduced errors.

6. Sustainability Assessment: SolidWorks Sustainability offers tools to evaluate the environmental impact of designs. Users can assess factors such as energy consumption and carbon footprint, aligning with the growing emphasis on sustainable and eco-friendly product development.

7. API (Application Programming Interface): For users seeking customization and automation, SolidWorks provides a robust API. This allows developers to create custom applications, automate repetitive tasks, and integrate SolidWorks into broader workflows.

8. 3D Printing Integration: SolidWorks supports 3D printing by providing tools to prepare and optimize designs for additive manufacturing. This feature aligns with the evolving trends in manufacturing and facilitates the seamless transition from digital design to physical prototypes.

SolidWorks, with its rich set of features, continually evolves to meet the demands of an ever-changing design landscape. Whether it's the precision of parametric modeling, the realism of rendering, or the integration of advanced simulation tools, SolidWorks remains at the forefront of 3D design, empowering innovators to turn their ideas into reality with efficiency and precision.

Our sketch in solid Works:

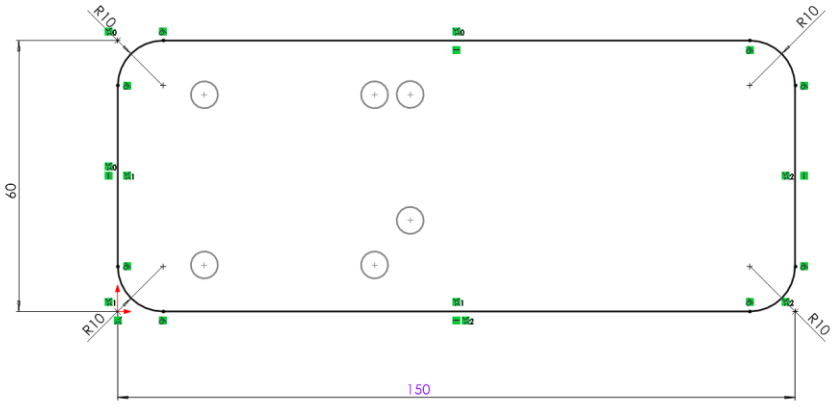
Creating a plate in SolidWorks involves several steps, including sketching, extruding, and adding any necessary details. Here's a step-by-step procedure for creating a plate with dimensions 16 x 7:

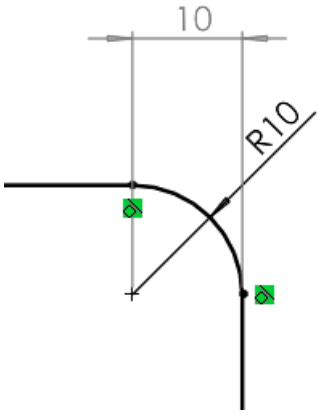
1. **Open SolidWorks:**
   - Launch SolidWorks on your computer.
2. **Create a New Part:**
   - Start a new part document.
3. **Sketching:**
   - Click on the "Sketch" tab in the top menu.
4. **Create a Rectangle:**
   - Select the "Rectangle" tool from the sketch toolbar.
   - Choose the "Corner Rectangle" option.
   - Click at the origin (0,0) and drag the cursor to create a rectangle.
   - Enter the dimensions in the respective boxes. For a 16 x 7 plate, enter 16 for the length and 7 for the width.
   - Click on "OK" or press Enter to complete the sketch.
5. **Exit Sketch Mode:**
   - Click on the "Exit Sketch" button to exit sketch mode.
6. **Extrude:**
   - Click on the "Features" tab in the top menu.
   - Select "Extrude Boss/Base."
   - Enter the desired thickness for the plate. If you want a solid plate, simply enter the thickness, or you can choose the "Mid Plane" option to extrude in both directions.
   - Click "OK" to complete the extrusion.
7. **Optional: Add Fillets or Chamfers (if desired):**
   - If you want to add rounded edges, click on the "Fillet" tool or "Chamfer" tool in the top menu.
   - Select the edges you want to fillet or chamfer and specify the dimensions.
8. **Save Your Part:**
   - Click on "File" and then "Save" to save your part in a desired location.
9. **Optional: Add Additional Features:**
   - You can add holes, patterns, or any other features to the plate using the various sketch and feature tools available in SolidWorks.
10. **Finalize:**

- Review your part and make any additional modifications if needed.

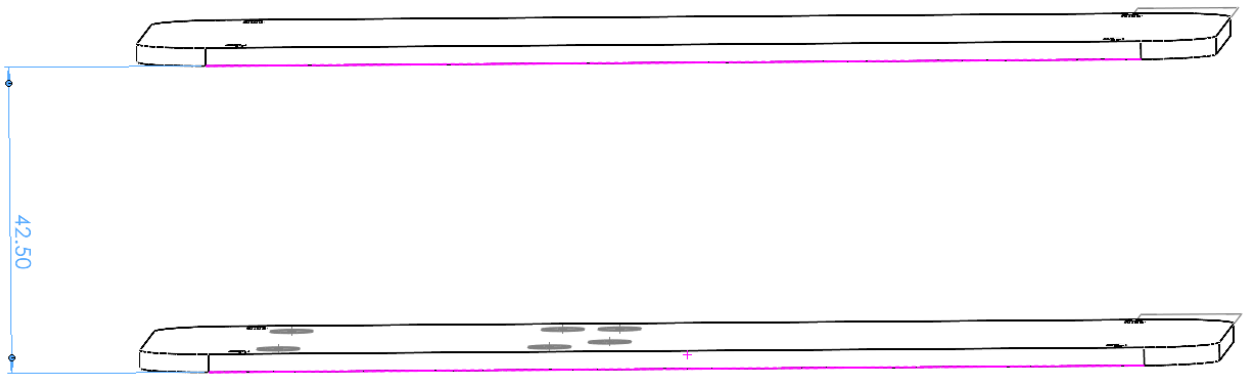
- Once satisfied, you can use the "Save As" option to save different versions or formats of your part.

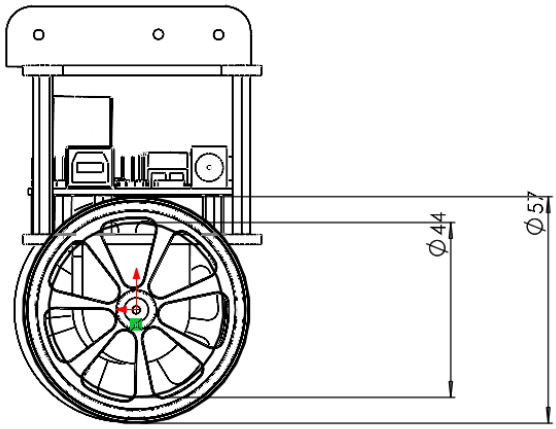
By following these steps, you should be able to create a simple rectangular plate with dimensions 16 x 7 in SolidWorks. Feel free to explore additional features and tools to enhance your design based on your specific project requiremen





The height of each plates should be such that no collision happened between parts and plates





The plates are 150 x 7. And we have implemented it on the solid works sketch, the radius is 5.7 cm since we couldn't find a wheel of more radius in the market. Every mechanism is shown in the solid works shape and the main files are also attached with this file.

About Actuators

The 6-volt DC motor : Having been bought from a local store , it didn't come with a datasheet so little is known about its specification. For our use we state that its nominal voltage is approximately 6volts and it's coupled with 1024-pulse encoder which then is controled with a driver. We program it such that its speed could be controled , while rotating in one direction

Inertial Measurement Unit (IMU): Unraveling the Mechanism of Motion Sensing

The Inertial Measurement Unit (IMU) is a sophisticated sensor module that plays a pivotal role in measuring and understanding the orientation, velocity, and gravitational forces acting on a moving object. Comprising accelerometers and gyroscopes, IMUs employ a combination of these sensors to provide comprehensive insights into the dynamics of motion.

1. Accelerometers: Accelerometers within an IMU measure linear acceleration along specific axes.

They operate based on the principles of inertial forces, where changes in velocity result in proportional forces acting on internal structures. By detecting these forces, accelerometers can determine the rate at which an object is accelerating or decelerating in a particular direction. In essence, they sense changes in velocity and gravitational forces, crucial for applications ranging from navigation systems to detecting tilt and motion in consumer electronics.

2. Gyroscopes: Gyroscopes, on the other hand, are responsible for measuring angular velocity or

the rate of rotation around specific axes. They leverage the concept of angular momentum, where the conservation of rotational motion is utilized to ascertain the object's orientation. Gyroscopes help track the rate at which an object is rotating, aiding in tasks such as image stabilization in cameras, navigation systems, and, significantly, the control of autonomous vehicles and drones.

3. Sensor Fusion: The true power of IMUs lies in sensor fusion, a process where data from accel-

erometers and gyroscopes are combined to provide a more accurate and reliable representation of an object's motion. This fusion compensates for the limitations inherent in each sensor type. For instance, accelerometers may suffer from drift over time, while gyroscopes may experience errors in long-term measurements. By intelligently integrating data from both sensors, IMUs can offer a more precise understanding of complex motions, such as those encountered in robotics, virtual reality applications, and aerospace systems.

4. Applications: IMUs find applications in diverse fields, ranging from smartphones and wearable

devices for gesture recognition to inertial navigation systems guiding spacecraft. In the context of robotics, IMUs are crucial for maintaining stability in dynamic environments, such as in self-balancing robots, where real-time data on orientation and acceleration are essential for responsive control systems.

In conclusion, IMUs are instrumental in deciphering the intricacies of motion through the collaborative efforts of accelerometers and gyroscopes. Their ability to provide real-time and accurate

data on the dynamics of movement fuels innovations across industries, contributing to advancements in navigation, robotics, and various emerging technologies.

Inertial Measurement Units (IMUs) use a process called sensor fusion to integrate data from accelerometers and gyroscopes to determine the roll, pitch, and yaw of an object. Each of these Euler angles represents a specific orientation or rotation of the object in three-dimensional space.

Here's how the integration of data from accelerometers and gyroscopes works to obtain roll, pitch, and yaw:

1. **Accelerometer Data for Roll and Pitch:**
    - Accelerometers measure the linear acceleration along specific axes, including the effects of gravity. When an object is at rest on a level surface, the accelerometer measures the gravitational acceleration. Tilt or movement in any direction introduces changes in the gravitational component sensed by the accelerometers.
    - By processing these changes, the system can calculate the roll and pitch angles. The arctangent of the ratio of acceleration in the perpendicular axes to the gravitational acceleration provides the roll and pitch angles.
2. **Gyroscope Data for Rate of Change:**
    - Gyroscopes measure the rate of angular velocity or the rate of rotation around specific axes. They provide continuous data indicating how fast the object is rotating.
    - Integration of gyroscope data over time yields the angle of rotation. However, gyroscope data is subject to drift, causing cumulative errors over time. To address this, gyroscope data is commonly used for short-term calculations and combined with accelerometer data to correct for long-term drift.
3. **Complementary Filter or Kalman Filter:**
    - The integration of accelerometer and gyroscope data is often done using a complementary filter or a Kalman filter.
    - A complementary filter blends accelerometer and gyroscope data in a way that takes advantage of the low-frequency accuracy of accelerometers for tilt measurements and the high-frequency accuracy of gyroscopes for capturing rapid changes.
    - A Kalman filter, a more sophisticated approach, is an algorithm that optimally fuses accelerometer and gyroscope data, minimizing errors and providing accurate estimates of orientation.
4. **Conversion to Roll, Pitch, and Yaw:**
    - The obtained data is then converted to the Euler angles: roll (rotation around the longitudinal axis), pitch (rotation around the lateral axis), and yaw (rotation around the vertical axis).

By combining the information from accelerometers and gyroscopes through sensor fusion techniques, IMUs can provide a more accurate and robust estimation of an object's orientation in three-dimensional space, allowing for precise control and navigation in applications such as robotics, drones, and virtual reality systems.

Handling noise is a critical aspect of IMU (Inertial Measurement Unit) data processing because noise can introduce inaccuracies and affect the reliability of orientation and motion measurements. IMUs often encounter different types of noise, including sensor noise, environmental interference, and electronic noise. Here are several strategies employed to handle noise in IMU data:

1. **Filtering Techniques:**
    - **Low-Pass Filters:** Low-pass filters are commonly used to attenuate high-frequency noise while preserving the lower-frequency signals that represent the actual motion. These filters can help eliminate high-frequency components that may arise from sensor noise or vibrations.
    - **Kalman Filtering:** Kalman filters are recursive algorithms that combine information from different sensors while considering the statistical properties of the noise. Kalman filters are effective in providing optimal estimates by continuously updating and refining the sensor measurements.
2. **Sensor Calibration:**
    - **Bias Correction:** IMUs may have biases in sensor readings, contributing to errors and noise. Calibration procedures, including bias correction, are essential to mitigate these inaccuracies. Calibration involves determining and compensating for biases and other systematic errors in the sensor data.
    - **Scale Factor Correction:** Calibration may also include correcting scale factors in sensor measurements, ensuring that the sensor's sensitivity is consistent across different axes.
3. **Temperature Compensation:**
    - Temperature variations can impact the accuracy of IMU sensors. Implementing temperature compensation algorithms helps correct for thermal effects, ensuring that the IMU remains accurate across different environmental conditions.
4. **Sensor Fusion:**
    - Combining data from multiple sensors, such as accelerometers and gyroscopes, through sensor fusion techniques can enhance accuracy and reduce the impact of

noise. Algorithms like complementary filters and Kalman filters are used to intelligently merge data from different sensors, taking advantage of the strengths of each sensor type.

5. **Signal Averaging:**
    - Averaging sensor readings over time can help smooth out noise. This technique is particularly useful for reducing short-term fluctuations and improving the signal-to-noise ratio.
6. **Advanced Algorithms:**
    - Implementing more advanced signal processing algorithms, such as wavelet transforms or advanced filtering methods, can be effective in isolating and removing noise components.
7. **Outlier Rejection:**
    - Identifying and discarding outlier data points can help mitigate the impact of sporadic noise spikes. Techniques such as statistical analysis or threshold-based rejection can be applied to filter out unreliable data.
8. **Continuous Monitoring and Updating:**
    - IMU systems often include continuous monitoring and updating mechanisms. Real-time assessment of sensor performance allows for adaptive adjustments and recalibration, ensuring that the system maintains accuracy over time.

By employing a combination of these techniques, IMUs can effectively handle noise and provide more accurate and reliable motion and orientation data in various applications, ranging from navigation systems to robotics and virtual reality.
An Inertial Measurement Unit (IMU) is an electronic device that combines multiple sensors to measure and report an object's specific force, angular rate, and sometimes the magnetic field surrounding it. IMUs are crucial components in various applications, including robotics, aerospace, navigation systems, virtual reality, and motion tracking. Here are the key components and functionalities of an IMU:

1. Accelerometers:

- Accelerometers measure the acceleration experienced by an object along its three spatial axes (X, Y, and Z). They detect changes in velocity, including the effects of gravity. Accelerometers are fundamental for understanding linear motion, tilt, and orientation.

2. Gyroscopes:

- Gyroscopes, or gyros, measure the rate of rotation around the three spatial axes. They provide information about angular velocity, allowing the determination of rotational motion and changes in orientation. Gyroscopes are essential for capturing the object's angular dynamics.

3. Magnetometers (Optional):

- Some IMUs incorporate magnetometers to measure the strength and direction of the magnetic field around the object. Magnetometers aid in determining the object's orientation relative to the Earth's magnetic field. This is particularly useful in applications where absolute orientation is required, such as navigation systems.

4. Microcontroller/Processor:

- The microcontroller or processor is the brain of the IMU. It processes data from the accelerometers, gyroscopes, and, if present, magnetometers. The microcontroller executes algorithms for sensor fusion, calibration, and filtering to obtain accurate and reliable motion and orientation information.

5. Sensor Fusion Algorithms:

- Sensor fusion is a crucial aspect of IMU functionality. It involves combining data from multiple sensors to improve accuracy and reduce errors. Popular sensor fusion algorithms include Kalman filters, complementary filters, and Mahony filters. These algorithms intelligently blend accelerometer and gyroscope data to obtain a more accurate estimation of orientation.

6. Calibration:

- Calibration is essential to compensate for biases, offsets, and scale factor errors in sensor readings. IMUs undergo calibration procedures during manufacturing, and users may perform additional calibration to enhance accuracy for specific applications.

7. Output Formats:

- IMUs provide output data in various formats, including Euler angles (roll, pitch, yaw), quaternion representations, or raw sensor data. The choice of output format depends on the application and user preferences.

8. Applications:

- IMUs find applications in diverse fields, including robotics for navigation and control, aerospace for flight control systems, virtual reality for tracking head movements, wearable devices for activity monitoring, and automotive systems for stability control.

9. MEMS Technology:

- Many modern IMUs use Micro-Electro-Mechanical Systems (MEMS) technology. MEMS sensors are miniaturized and integrate accelerometers and gyroscopes on a single chip. MEMS-based IMUs are compact, lightweight, and cost-effective.

IMUs play a crucial role in enabling precise motion sensing and orientation tracking in a wide range of technological applications. Their versatility and accuracy make them indispensable in modern electronics and control systems.

18650 Batteries: Powering Innovation with Compact Energy

The 18650 battery is a cylindrical rechargeable lithium-ion battery that has become a standard power source in a wide array of electronic devices. Its name is derived from its dimensions: 18mm in diameter and 65mm in length. Originally developed for use in laptops, these batteries have found applications in countless devices due to their high energy density, reliability, and rechargeable nature.

Key Features: 18650 batteries are characterized by several key features that contribute to their popularity:

1. **High Energy Density:** 18650 batteries are known for their impressive energy density, meaning they can store a significant amount of energy in a relatively compact form. This makes them ideal for applications where space and weight considerations are crucial.
2. **Rechargeable:** Unlike disposable batteries, 18650 batteries are rechargeable. This feature not only contributes to cost savings but also aligns with the growing emphasis on sustainability and reducing environmental impact.
3. **Voltage and Capacity:** Typically, 18650 batteries have a nominal voltage of 3.7 volts. The capacity varies, with common capacities ranging from around 2000mAh to 3500mAh. Higher capacity batteries provide more extended usage before requiring a recharge.

4. **Versatility:** 18650 batteries are used in an extensive range of applications, including laptops, flashlights, power tools, electric vehicles, and various consumer electronics. Their versatility stems from their ability to deliver a consistent and reliable power supply.
5. **Standardized Form Factor:** The 18650 form factor has become a standard in the battery industry. This standardization ensures compatibility with a wide range of devices and charging systems, contributing to their widespread adoption.

Applications: 18650 batteries power a diverse range of devices and systems:

1. **Consumer Electronics:** They are commonly used in laptops, digital cameras, flashlights, and portable electronic devices.
2. **Power Tools:** 18650 batteries provide the energy needed for cordless power tools, offering a balance between power and weight.
3. **Electric Vehicles:** In some electric vehicles, a large number of 18650 batteries are combined to form a battery pack, providing the necessary energy for propulsion.
4. **Renewable Energy Storage:** 18650 batteries are also used in renewable energy storage systems, such as solar power storage, providing a reliable power source during periods of low sunlight.
5. **Vaping Devices:** The vaping industry extensively employs 18650 batteries to power electronic cigarettes and vaping devices.

Considerations: While 18650 batteries offer numerous advantages, users should be aware of safety considerations, including proper charging procedures, avoiding over-discharge, and ensuring compatibility with the intended device.

In summary, 18650 batteries have become integral to our daily lives, powering an extensive range of electronic devices. Their compact size, rechargeable nature, and high energy density make them a go-to choice for applications demanding reliable and efficient power sources.
Tuning the PID (Proportional-Integral-Derivative) controller for a self-balancing robot is a crucial process to achieve stability and responsiveness. The PID controller is widely used in self-balancing systems to maintain the desired orientation by adjusting the motor speeds. Tuning involves adjusting the three parameters: proportional gain (P), integral gain (I), and derivative gain (D). Here's a comprehensive guide examining each possibility:

1. Proportional Gain (P):

- **Effect:** The proportional term determines how much the robot should lean in response to the error (the difference between the desired and actual angles).
- **Procedure:** Start with a low P value and increase it gradually until the robot starts oscillating. Once oscillation is observed, reduce the P value slightly to find the critical point just before oscillation.

2. Integral Gain (I):

- **Effect:** The integral term addresses long-term errors, particularly those caused by biases or constant disturbances.
- **Procedure:** Begin with a low I value and gradually increase it until steady-state errors (constant leaning) are minimized. Be cautious, as too much integral gain may introduce instability or overshooting.

3. Derivative Gain (D):

- **Effect:** The derivative term helps dampen the system's response and reduce overshooting.
- **Procedure:** Start with a D value of zero and gradually increase it until the robot's response improves. D can help in reducing oscillations and overshooting caused by sudden changes in the system.

4. Ziegler-Nichols Method:

- **Procedure:** Use the Ziegler-Nichols method, a systematic approach for PID tuning. Start by setting all gains to zero and gradually increase the P value until the system starts oscillating. The critical P value and the oscillation period can be used to calculate the ultimate gain (Ku) and ultimate period (Tu). Then, set P = 0.6Ku, I = 2Tu, and D = Tu/8.

5. Trial and Error:

- **Procedure:** Conduct trial and error tuning by observing the robot's response to different gain values. Adjust one parameter at a time while keeping others constant. This method requires patience and experimentation.

6. Auto-Tuning Algorithms:

- **Procedure:** Some PID controllers have auto-tuning algorithms that adjust the gains based on the system's response. While convenient, they may not always provide optimal results and might need manual refinement.

7. Iterative Refinement:

- **Procedure:** Gradually refine the gains based on real-world testing. Observe the robot's behavior in various conditions and make incremental adjustments to achieve the desired balance between stability and responsiveness.

Considerations:

- **Physical Properties:** The robot's weight distribution, wheel size, and motor characteristics significantly impact PID tuning. Adjustments may be needed when these physical properties change.
- **Environment:** Different surfaces or terrains may require adjustments to the PID gains. The robot's behavior may vary on smooth floors compared to rough surfaces.

Conclusion: PID tuning for a self-balancing robot is an iterative process that involves a balance between stability and responsiveness. It requires a deep understanding of the robot's dynamics, experimentation, and patience. Continuous testing and refinement based on real-world performance are essential to achieve optimal results. Consider the robot's physical properties, environmental conditions, and the specific requirements of your application. By exploring each possibility and carefully observing the robot's behavior, you can fine-tune the PID controller to enable precise and stable self-balancing.

Low-pass Filter :

Low Pass Filter in the Context of Self-Balancing Robots:

A Low Pass Filter (LPF) is a fundamental component in the control systems of self-balancing robots. Its primary function is to attenuate high-frequency noise or rapid changes in the sensor data, allowing the system to focus on the slower, meaningful signals related to the robot's orientation and motion. In the context of self-balancing robots, where precise control is crucial, the LPF aids in smoothing out sensor readings, reducing the impact of vibrations and high-frequency disturbances.

Transfer Function of a Low Pass Filter:

The transfer function of an ideal first-order low pass filter can be represented in the frequency domain as:

$$H(s)=\frac{1}{1+sT_1}$$

Here,

- $H(s)$ is the transfer function.
- s is the complex frequency variable.
- T is the time constant of the filter.

The time constant (T) determines how quickly the filter responds to changes in the input signal.

A larger time constant results in a slower response, effectively allowing lower-frequency components to pass through the filter while attenuating higher frequencies.

In the time domain, the transfer function can be expressed as:

$$\text{Output}(t)=\int \frac{1}{T_1}\cdot \text{Input}(t)\,dt$$

In practical terms, implementing a low pass filter involves discrete-time processing, especially in digital control systems. A common approach is to use a digital Butterworth filter due to its smooth

frequency response and straightforward implementation. The transfer function of a discrete-time Butterworth low pass filter is given by:

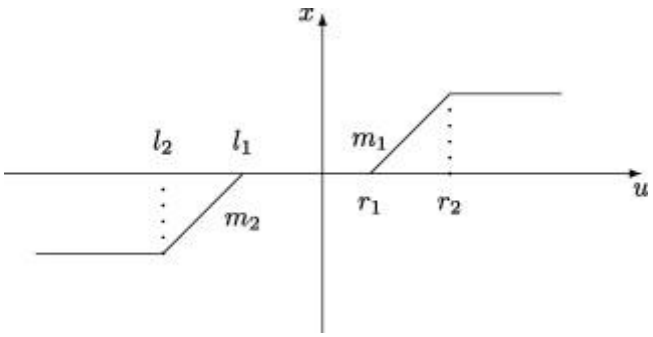
$$H(z)=\frac{K}{1+z^{-1}}$$

Here, z is the complex variable in the Z-domain, and K is the gain factor.

Application in Self-Balancing Robots:

In self-balancing robots, accelerometers and gyroscopes provide essential data for determining the robot's orientation. However, these sensors may introduce noise or rapid fluctuations, especially in dynamic environments. The LPF, integrated into the control system, helps mitigate these effects by selectively allowing the lower-frequency components associated with the robot's tilt and motion to pass through while suppressing high-frequency noise.

By incorporating a low pass filter into the sensor data processing pipeline, self-balancing robots can achieve more stable and accurate control. This is particularly important when using the sensor data as feedback for a PID controller, as the filtered signal contributes to smoother and more reliable control actions, ultimately enhancing the robot's ability to maintain balance in real-world conditions.



Understanding Motor Dead Zone in Control Systems:

In control systems, a motor dead zone refers to a specific region around the zero-input position where no motion or response occurs despite the presence of an input signal.

This dead zone is a phenomenon commonly observed in mechanical systems and can have implications for system performance, stability, and control accuracy.

1. **Definition and Characteristics:

- A motor dead zone typically occurs due to mechanical slack, friction, or other nonlinearities in the system. It is an interval of input values around the zero-point where the motor or actuator does not respond, causing a delay or lack of motion even when a control signal is applied.
- Characteristics of a dead zone may include a delayed start, sudden jumps in response, or a lack of precision in motion.

2. **Causes of Motor Dead Zone:

- **Friction and Mechanical Slack:** In mechanical systems, components like gears, couplings, and linkages may introduce friction or slack, creating a dead zone where the motor must overcome inertia or resistance before initiating motion.
- **Backlash:** Dead zones can arise from backlash in gears or other mechanical linkages, causing a delay between changes in input and corresponding output motion.
- **Nonlinearities:** Nonlinearities in motor characteristics, such as hysteresis or saturation, can contribute to dead zones.

3. **Effects on Control Systems:

- **Impact on Stability:** Motor dead zones can impact the stability of a control system, especially in applications where precise and rapid responses are critical. The presence of dead zones can lead to overshooting, oscillations, or difficulties in achieving the desired setpoint.
- **Control Accuracy:** Dead zones may result in reduced accuracy in control applications, as the system may not respond as expected, leading to errors in positioning or tracking tasks.
- **Complicating Controller Design:** Dead zones complicate the design of control algorithms, as controllers need to account for these nonlinearities to ensure robust and effective performance.

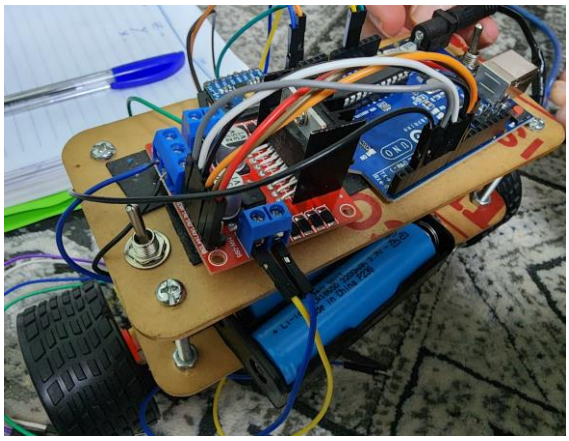
4. **Mitigation Strategies:

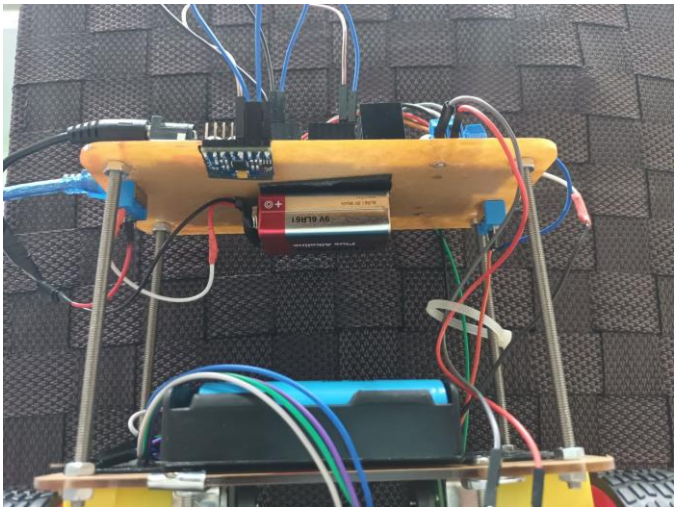
- **Compensation:** Designing control algorithms that compensate for the presence of dead zones can help mitigate their effects. This may involve introducing lead or lag compensators in the control system.
- **Higher Resolution Sensors:** Upgrading sensors to higher resolution devices can enhance the system's ability to detect and respond to small changes, minimizing the impact of dead zones.
- **Mechanical Improvements:** Addressing mechanical issues such as reducing friction, backlash, or slack in the system can help minimize dead zones.

5. **Practical Examples:

- **Robotics:** In robotic systems, dead zones can affect the accuracy of manipulator arms, leading to imprecise movements or positioning errors.
- **Automotive Systems:** Dead zones in steering mechanisms or throttle control can impact the responsiveness and handling of vehicles.
- **Industrial Automation:** Motor dead zones in manufacturing equipment may affect the precision and repeatability of processes.

In conclusion, understanding and mitigating motor dead zones is crucial for the effective design and operation of control systems. Engineers and control system designers need to consider the specific characteristics and causes of dead zones in a particular application to implement appropriate measures for improving stability, accuracy, and overall system performance.





The coding :

```
/***********PID************/
#include <PID_v2.h>

double kp = 8; // To be Tuned
double ki = 0;  // To be Tuned
double kd = 0; // To be Tuned
double filtered_value = 0.0;
double setpoint, input, output; // PID Variables
double output_mod;
double output_prev = 0.0;
double error;
PID_v2 pid(kp, ki, kd, DIRECT); // PID Setup
```

```
/********** L298N **********/
#include <L298N.h>

// Pin Definition:
const unsigned int EN_A = 3;
const unsigned int IN1_A = 5;
const unsigned int IN2_A = 6;

const unsigned int IN1_B = 7;
const unsigned int IN2_B = 8;
const unsigned int EN_B = 9;

// Create motor instances
L298N rightMotor(EN_A, IN1_A, IN2_A);
L298N leftMotor(EN_B, IN1_B, IN2_B);

// Motor speeds
int speedLeft = 0;
int speedRight = 0;
```

```
/***********MPU************/
#include <I2Cdev.h>
#include <MPU6050_6Axis_MotionApps20.h>
#include <Wire.h>

MPU6050 mpu;
#define OUTPUT_READABLE_YAWPITCHROLL

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

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

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

float pitch;
long velocity;

float trimPot;
float trimAngle;

int IMUdataReady = 0;
```

```
void readIMUData() {
    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet
        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // Display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            pitch = (ypr[1] * 180/M_PI); // Adjust to degrees
            setpoint = 0.0;
            input = pitch;

            //Filter pitch using lowpass filter
            filtered_value = lowPassFilter(input,output_prev);
        #endif
        // Blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

```
void processPitch() {
    pitch = (ypr[1] * 180/M_PI); // Adjust to degrees
    setpoint = 0.0;
    input = pitch;
}

void runPIDController() {
    const double output = pid.Run(input);

    if (output > -10 && output < 10) {
        output_mod = 0;
    } else {
        output_mod = output;
    }

    speedLeft = output;
    speedRight = output;
}
```

```
void moveMotors() {
    if (pitch > 25 || pitch < -25) { // Angle threshold
        leftMotor.setSpeed(0);
        rightMotor.setSpeed(0);
    } else {
        leftMotor.setSpeed(abs(speedLeft));
        rightMotor.setSpeed(abs(speedRight));
    }

    // Move motors
    if (speedLeft > 0) {
        leftMotor.forward();
    } else {
        leftMotor.backward();
    }

    if (speedRight > 0) {
        rightMotor.forward();
    } else {
        rightMotor.backward();
    }

    // Print some control info
    Serial.print("pitch: ");
    Serial.print(ypr[1] * 180/M_PI);
    Serial.print(" modified_pitch: ");
    Serial.print(filtered_value);
    Serial.print(" output: ");
    Serial.print(output);
    Serial.print(" output_mod: ");
    Serial.print(output_mod);
    Serial.println();
}
```

Chapter III (Simulink):

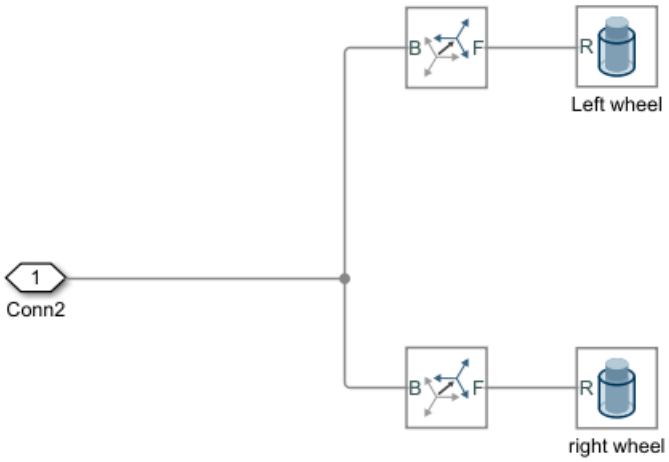
In this chapter , we implemented a simulation for our self balancing robot.

We use solver configuration which solves the system and mechanism configuration which I determined the gravity = G in +y direction.

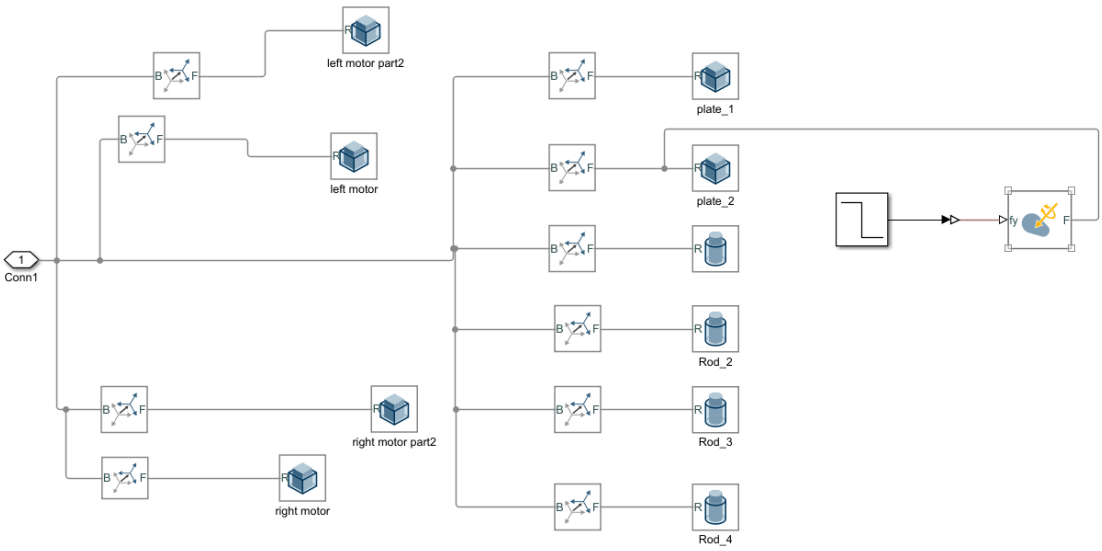
This configuration connects to prismatic joints which is the wheel models movements. And then after two rotations it will connects to our system.

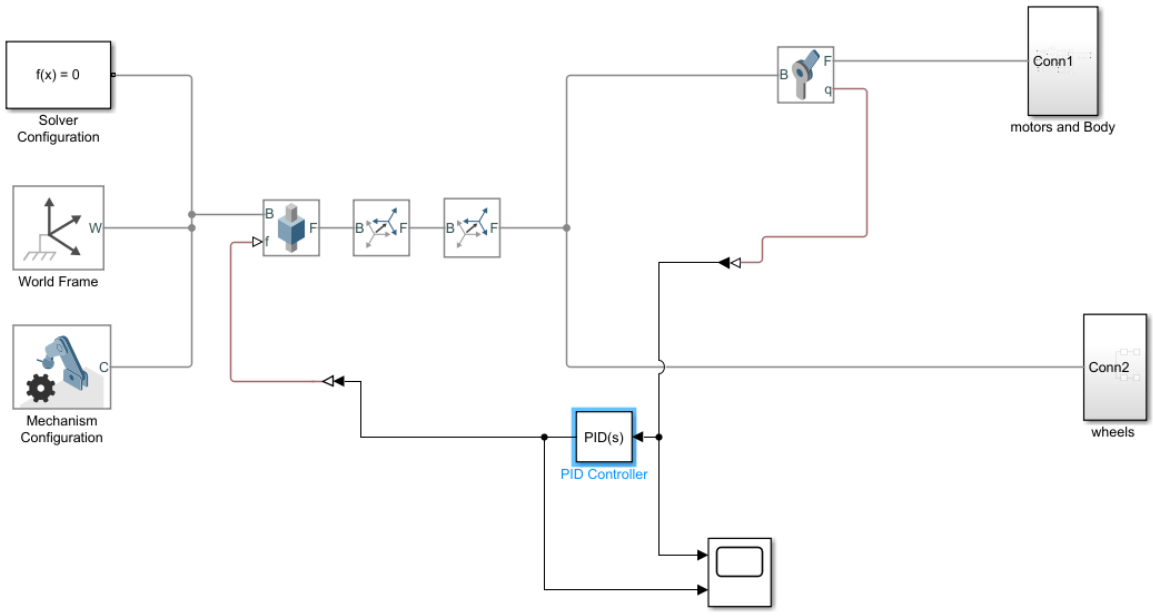
We use the rotations for having +y for gravity and Z axis for robot movement

As it can be seen we have two subsystems contain of body and motors and the wheel part. The revolute joint before motor and body subsystem connects these to part. We control our system with a pid controller and tuning these variables with MATLAB tuner tool.



In wheel subsystem we have the size and mass of two wheels which are identical. But their offset(position in the world frame). I extract these values from the raduis of the robot wheel , length of the robot wheel and their position in the robot.

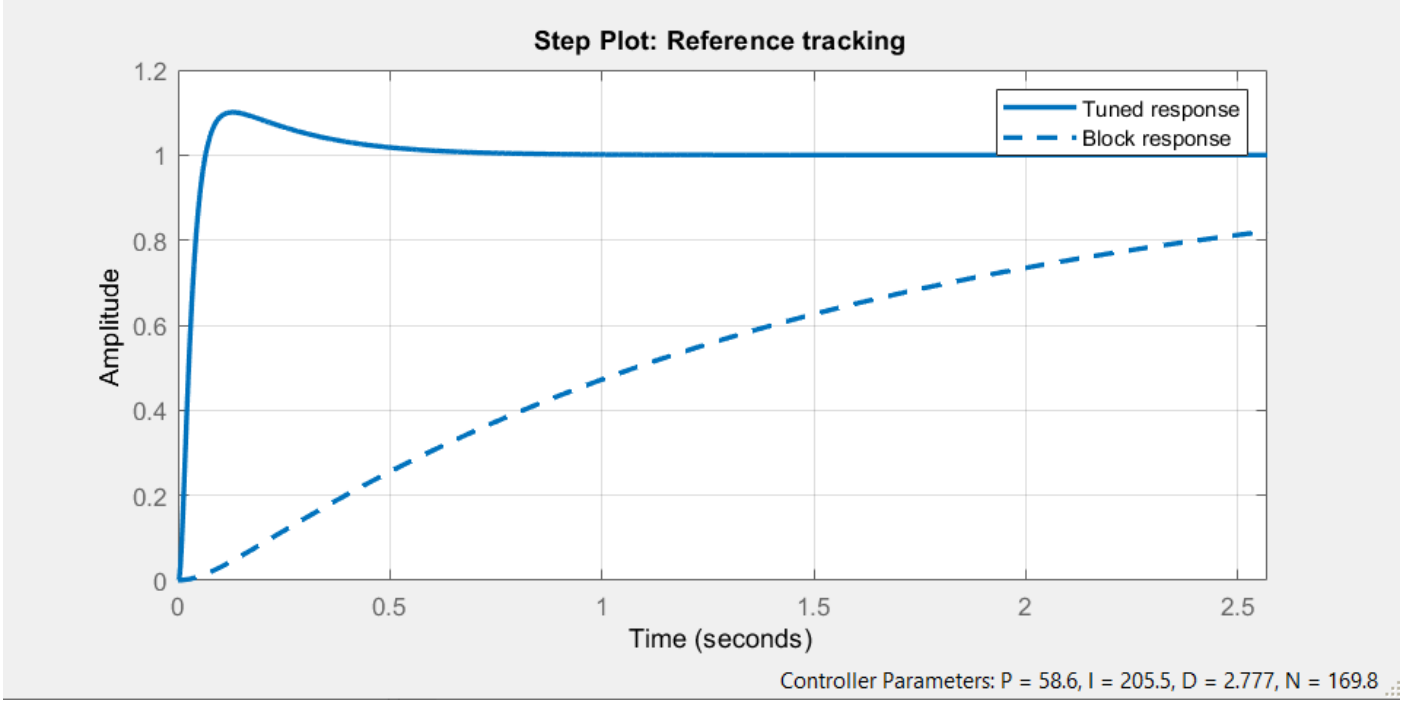

left motor part2
left motor
plate_1
plate_2
Rod_2
Rod_3
Rod_4
right motor part2
right motor
1
Conn1
fy
F



In the body part of the robot I implemented motors(as bricks) and the rods (as cylinders). As the privious part we defined off sets and size of each component In transforms and cylinder and brick solid block inside matlab. All of these are considered as a part in robot which connects the wheels via revolute joint.

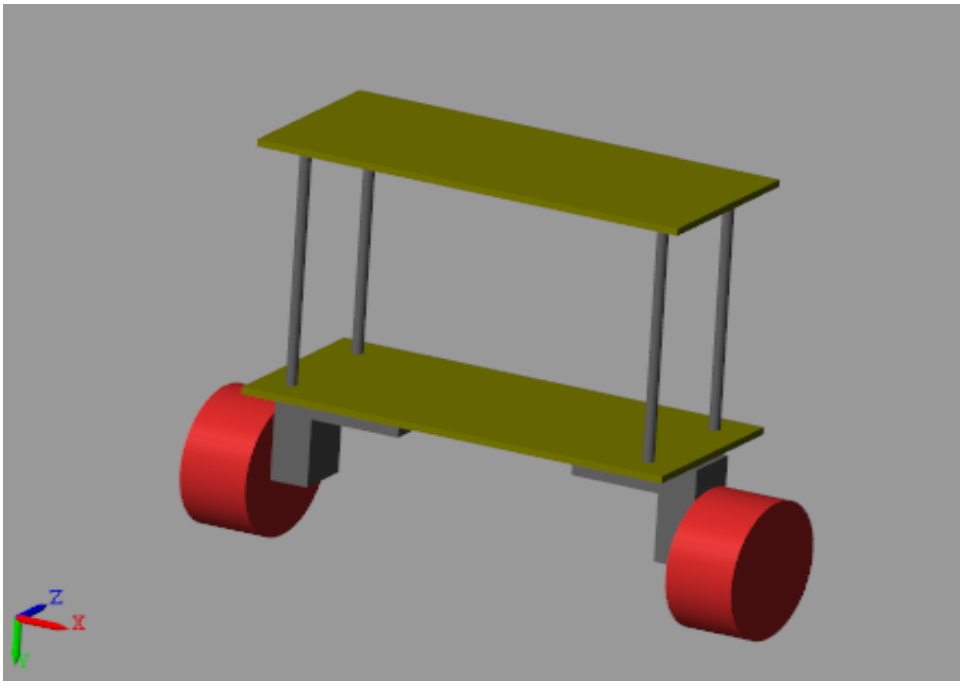
In this part we apply a forct(torqe) to the second plate(upper plate). This makes the robot unbalanced and it will try to balance itself using our PID controller. We use position mode in our revolute joint for getting position values as degrees and we apply a k = 180/3.14 for transforming degrees to rad and this will be our PID input. The out put of our PID in th force(N) which is applied to peismatic joint for moving the wheels. In this part we run matlab pid tuner in PID block and extracts the values of P , I , D and N. we apply these parameters and in the simulation see that model works perfectly because of the

idealistic conditions and well-choosed parametes. We use these parameters for our physical robot too. And with some changes applied to our physical model.



we can see the plot of tunned PID in PID tuner. And see the diffrence of our block response and tunned response.

Also we have the world frame which was presented in presntation with real time running.


Z
X
Y