Teensy BalanceBot Mk1

Course project in Embedded and autonomous systems

Nikola Totev

Software Engineering
Year 3

FN: 62271

Introduction

The following documentation will give the reader detailed information about the Teensy BalanceBot Mk1 project.

The goal of this project is to develop a balancing robot from the ground up. This means going through the full process of creating a product/project, everything from system analysis and development of the system to conducting tests to verify that all the objectives are met and a complete and detailed documentation of all the steps.

System Analysis

This section covers desired functionality of the system, how it can be modelled, what solutions currently exist and how they compare to the project being developed.

Desired Functionality

Overview

This is a relatively small project with limited time for development and testing, therefore the desired functionality has been reduced to the bare minimum. The features that have been chosen are enough to create an MVP (Minimum Viable Product) that can be used as a good starting point for future projects that require a balancing robot platform. Future development possibilities and improvements are discussed later in the documentation.

Possible Uses

Despite the small formfactor of the Teensy BalanceBot Mk1, it has a couple of use cases.

- The first use case is in education. It is simple enough to be an introduction to robotics for students, but it has plenty of possibilities for any future development.
- The second use case is in research. Robots are becoming very common in everyday life, and different situations require robots with different sets of features and form factors. With the limited feature set, BalanceBot Mk1 provides an excellent blank canvas for researchers to build upon.

Functionality

- **Angle control** - Angle control is the single most important feature for Teensy BalanceBot Mk1. Angle control goes well beyond staying in an upright position. Consider the situation where the robot is required to move from point A to point B. When repositioning, due to the nature of the robot, it must either lean forward or backward. This requires more consideration when developing the software, because the lean angles when moving are not constant, they depend on the speed of the robot.
- **Position hold** -This feature is not critical to the operation of the robot, but it is still part of the core functionality of the platform. Being able to hold a given position allows for easier testing and tuning of angle control, as well as any future features. As mentioned, this platform can find uses in education and research. In these cases, space is limited, and it is not desirable for the robot to drift away during testing.

System Model

Creating the model of a system is a crucial step in the development process. It allows for any problems to be found early before any time and materials have been put into physical prototypes. The model of a system also allows for simulations to be made. These simulations provide valuable insight into the way the system behaves in the real world.

In the case of BalanceBot Mk1 simulations are also a great way to determine how the system can be controlled and how it will behave based on the input.

Mathematical Model

Creating a mathematical model is the first step of creating a model of the system. This is because the movement of body can be described by a set of equations. If the derived equations are accurate enough, the simulation will also be a true enough to the real system.

Inverted pendulum model

Before the equations, the govern the system, it is important to look at the system in a simpler way. In this case it is important to see the FBD (Free Body Diagram) on (figure 1), the system is essentially an inverted pendulum on a cart.

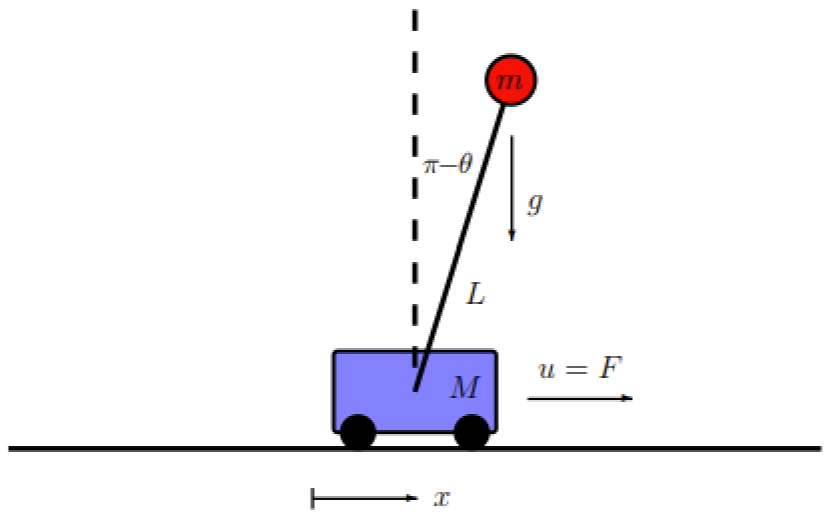


Figure 1 - System FBD (Steven L. Brunton, 2017)

The wheels of the robot can be thought of as the cart, and the rest of the body is the pendulum itself. It is not ultra-precise, but it is good enough for the goal of this project.

If greater accuracy is required, more time should be put into the modelling of the system.

Equations of motion

Equations of motion describe the behavior of a physical system in terms of its motion as a function of time (R.G. Lerner, 1991). Deriving such equations from a FBD is complicated and is outside the scope of this project, that is why that is why equations of motion in figure 2 that are already derived are used.

$$\dot{x} = v \tag{8.67a}$$

$$\dot{v} = \frac{-m^2L^2g\cos(\theta)\sin(\theta) + mL^2(mL\omega^2\sin(\theta) - \delta v) + mL^2u}{mL^2(M + m(1 - \cos(\theta)^2))} \tag{8.67b}$$

$$\dot{\theta} = \omega \tag{8.67c}$$

$$\dot{\omega} = \frac{(m + M)mgL\sin(\theta) - mL\cos(\theta)(mL\omega^2\sin(\theta) - \delta v) + mL\cos(\theta)u}{mL^2(M + m(1 - \cos(\theta)^2))} \tag{8.67d}$$

Figure 2 - Equations of motion (Steven L. Brunton, 2017, p. 352)

These non-linear equations of motion can be used to model the system and create a simulation. The details about creating the simulation are covered in subsequent sections.

Controller Options

When it comes to controllers there are a couple available options. This section is focused on examining three of the available options, analyzing pros and cons of each as well as ease of implementation.

System Controllability

This is a quintessential control theory problem. Before developing a system that requires control, it is important to check if it can be controlled. This check can be performed by calculating the rank of the controllability matrix of the system using the MATLAB command seen in Figure 3. In this case the rank should be 4 and if the code is run the rank will indeed be 4. This test confirms that the system can be controlled.

```
sys = ss(A, B, C, D);
ControlabilityMatrix = ctrb(sys);
rank(ControlabilityMatrix)
```

Figure 3 – ctrb() function in MATLAB

In figure 3 “sys” is a state-space representation of the system and the A & B matrices come from the equations of motion in figure 2, and the C and D matrices can be seen in figure 4.

```
C = [1,1,1,1];
D = zeros(size(C,1), size(B,2));
```

Figure 4 - Matrices C & D

The details about the state-space model, and mathematical theory are outside the scope of this documentation, but more information can be found in the book Data Driven Science & Engineering (Steven L. Brunton, 2017, p. 323).

Optimal Full state control - (LQR)

Optimal full state control is accomplished using a Linear Quadratic Regulator (LQR). With this method, the eigenvalues of the closed-loop system

$(A - BK)$ are manipulated through choice of a full-state feedback control law $u = -Kx$ (Steven L. Brunton, 2017, p. 343). For this method of control to be possible, the full state of the system must be available. This requires accurate sensors with little noise. Due to the available consumer-grade components discussed in future sections, this method is not a good choice for Teensy BalanceBot Mk1.

Full state estimation with (LQG)

Full state estimation is usually accomplished with a Kalman filter. The full-state estimate from the Kalman filter is generally used in conjunction with the full-state feedback control law from LQR, resulting in optimal sensor-based feedback (Steven L. Brunton, 2017, p. 350). This method requires more analysis to determine how observable the system is. It is possible that the full state of the system cannot be calculated from certain measurements. This method also relies on good sensor data and that is why for the first iteration of this project it is not implemented.

PID Controller

PID Controller is a control loop mechanism that uses feedback that to control systems that require continuously modulated control. It is widely used in industrial control systems and a variety of other applications such as robotics. It is the simplest of the three control methods discussed to implement on a microcontroller.

Such a controller does not require the highest accuracy from the sensors. Even noisy sensors can be good enough for a PID controller Another benefit PID controllers is that unlike LQR and LQG, creating a model that uses a PID controller is significantly easier.

Because of the easier implementation and less strict sensor requirements, a PID controller will be used to control Teensy BalanceBot Mk1.

The PID controller along with the model are made with Simulink because it provides the tools necessary to create the model of a system based on existing CAD models. The creation of the model, controller and the simulation are covered in greater detail in the following sections.

Modelling Software

This section discusses the software used to model the system. This includes CAD software as well as software used for modelling the system behavior and control.

Modelling software is an important part of the development of any system. It provides a cheap way of creating the system in a virtual environment. This virtual twin of the system can be used for future manufacturing – for example, the 3D models designed in CAD can be used for 3D printing or machining. The 3D models can also be imported into software like MATLAB & Simulink to simulate the behavior and control of the system.

Autodesk AutoCAD

CAD software is the most important modelling software for this project, this is because both the manufacturing stage and the system modelling stage in MATLAB depend on having a 3D model of the system.

The first reason Autodesk AutoCAD was chosen for this project as the sole designer of the system has experience working with the software. The second reason is that AutoCAD provides all the functionality required by the project.

MATLAB & Simulink

For creating the model of the system and to model the dynamics and control MATLAB with Simulink and Simscape is used. Other software products that have similar functionality exist such as Mathematica, but MATLAB is easier to use and has features designed specifically for system modelling.

MATLAB Model & Simulation

This section discusses the MATLAB model created in Simulink, the results from the model and the PID controller developed for the model.

Simulink Model

The Simulink model is divided into two parts. The first is the model of the robot, and the second are the PID controllers. The first part can be seen in figure 5, and the second in figure 6. . A larger version of the schematis can be found on the Github (Nikola, 2021) page of the project.

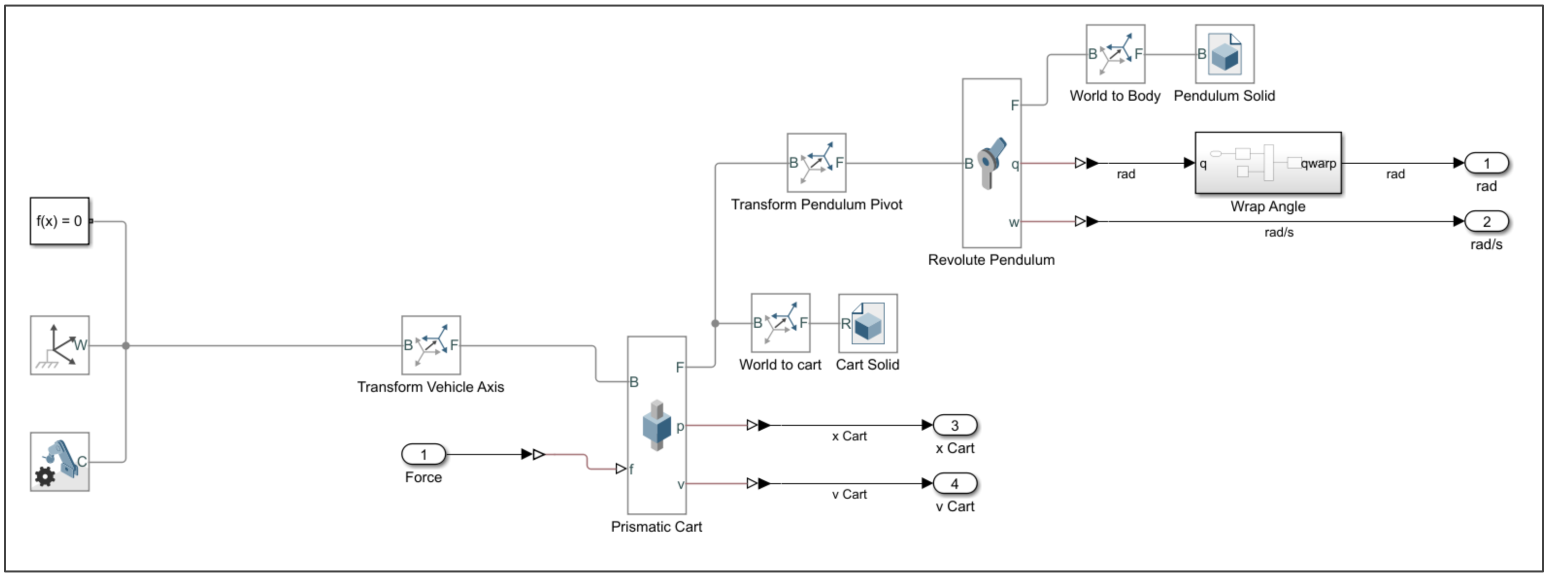


Figure 5 - Robot Model (no control)

Controller model

This section looks at the controller model, what kind of input it takes, how it is tuned and how the input is fed into the robot model.

PID Controller model

As seen on figure 6, there are 2 PID controllers. The inner loop is for the angle control and the outer loop is for position control. Both use the PID block available in Simulink.

The input for the angle controller is comprised of two things – setpoint and actual angle. The angle measurement comes from the revolute joint of the system. The setpoint can be toggled either to 0 or to the output of the position control.

The position control takes in a setpoint which in this case is a constant block with a value of 0 and the position of the robot, which in this case is the position output from the prismatic joint of the model. The output, as mentioned before, is fed as the setpoint to the angle control.

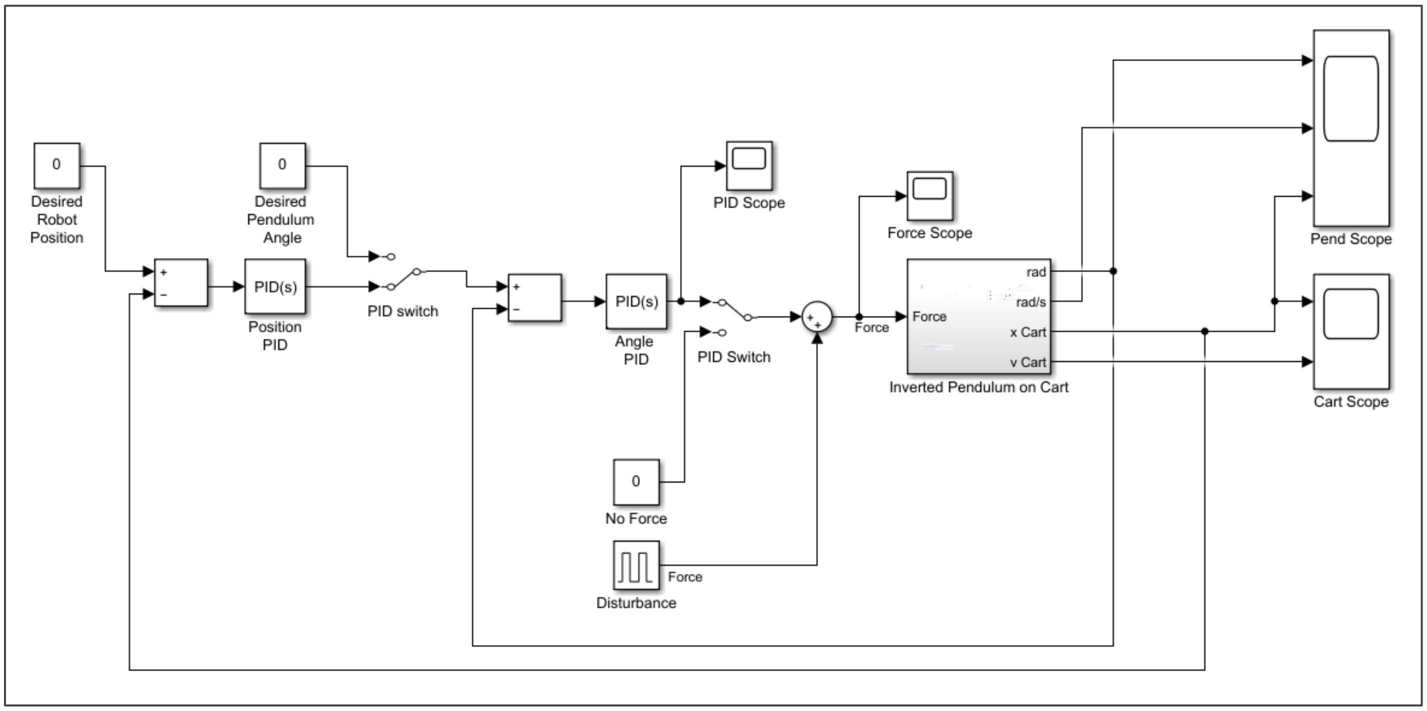
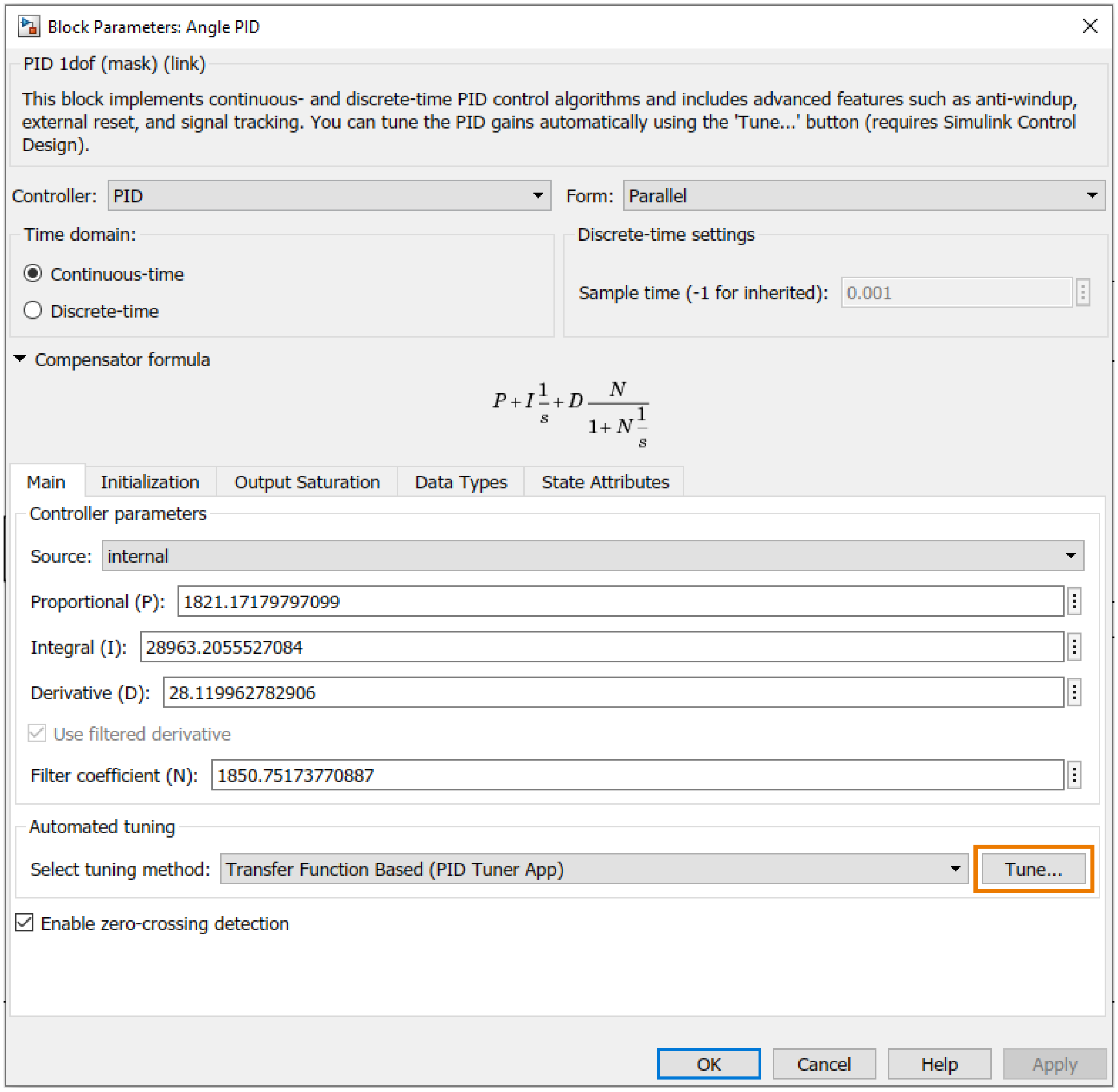


Figure 6 - PID Controllers

Automatic PID Tuning in MATLAB

If the PID controller block is double-clicked, the property window opens (See figure 7).



Block Parameters: Angle PID

PID 1dof (mask) (link)

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID   Form: Parallel

Time domain: [✓ Continuous-time] Discrete-time

Discrete-time settings: Sample time (-1 for inherited): 0.001

Compensator formula

$$P + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}$$

Main | Initialization | Output Saturation | Data Types | State Attributes

Controller parameters

Source: internal

Proportional (P): 1821.17179797099

Integral (I): 28963.2055527084

Derivative (D): 28.119962782906

☑ Use filtered derivative

Filter coefficient (N): 1850.75173770887

Automated tuning

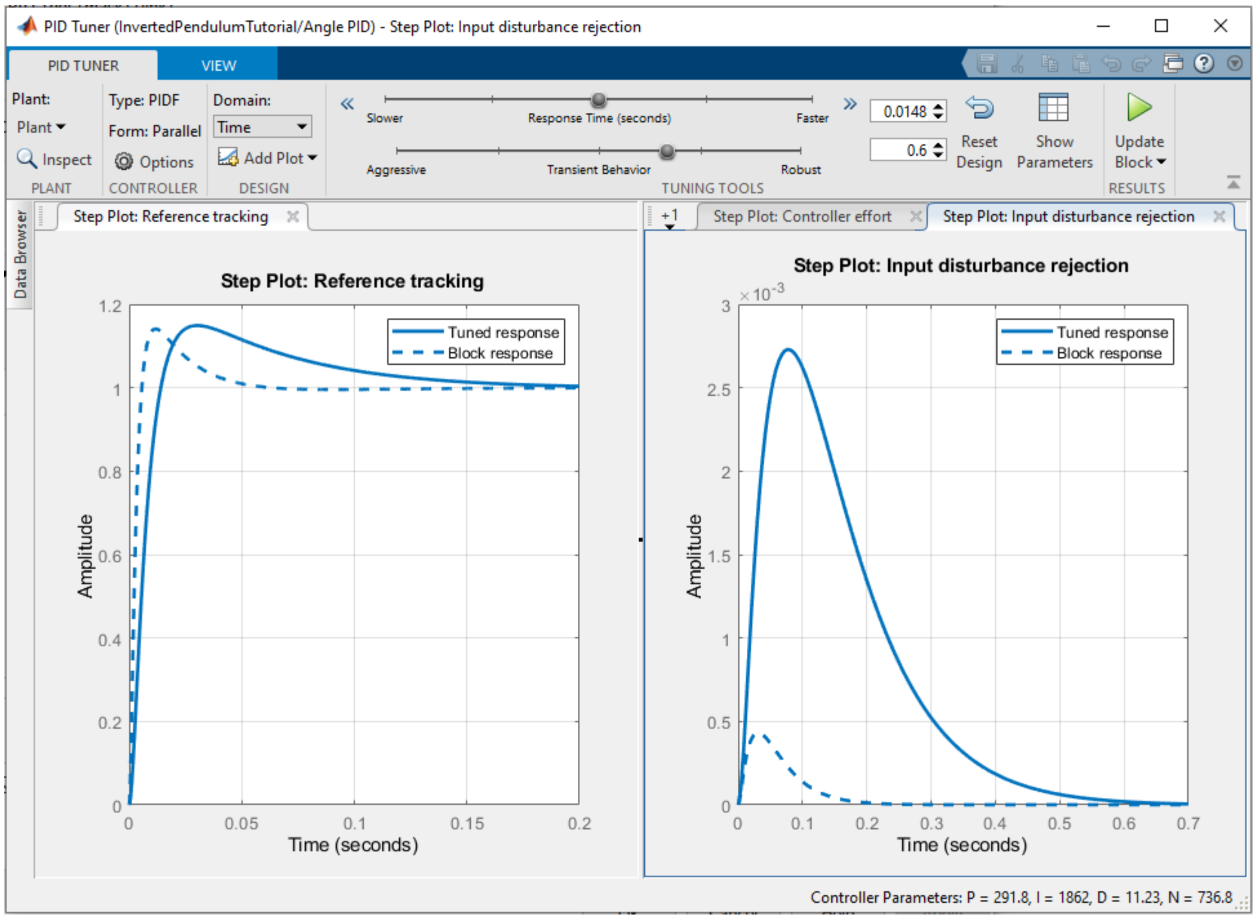
Select tuning method: Transfer Function Based (PID Tuner App)   Tune...

☑ Enable zero-crossing detection

Figure 7 - PID Controller Options

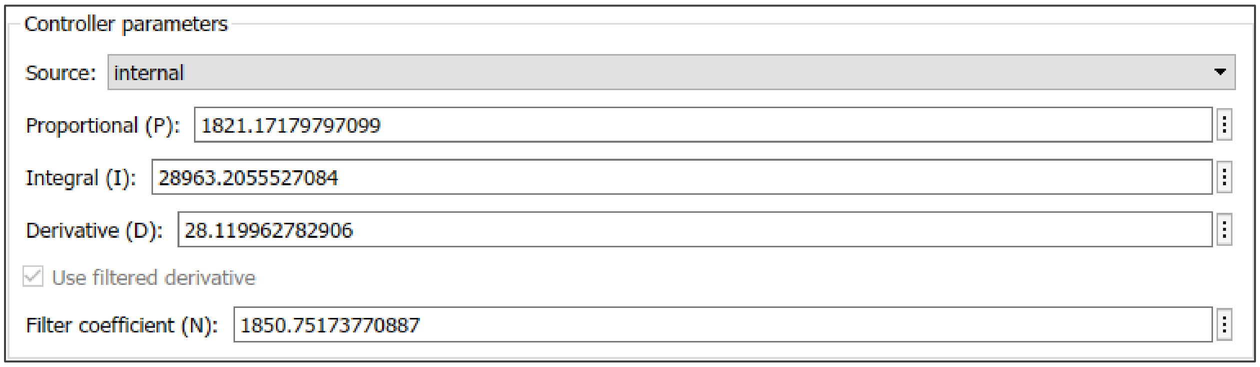
At the bottom right, there is a tune button. This is how the PID controller is tuned. Manual tuning is also possible, but it takes a considerable amount of time and it is not as accurate.

After pressing the “Tune” button, another window opens as seen on figure 8, and different parameters can be changed to modify the way the controller affects the system (how fast the system responds or how stable it is). After some testing, of different settings, on figure 9 you can see the final PID constants for the position and angle controllers, respectively.



Controller Parameters: P = 291.8, I = 1862, D = 11.23, N = 736.8

Figure 8 – Tuning window


Controller parameters

Source: internal

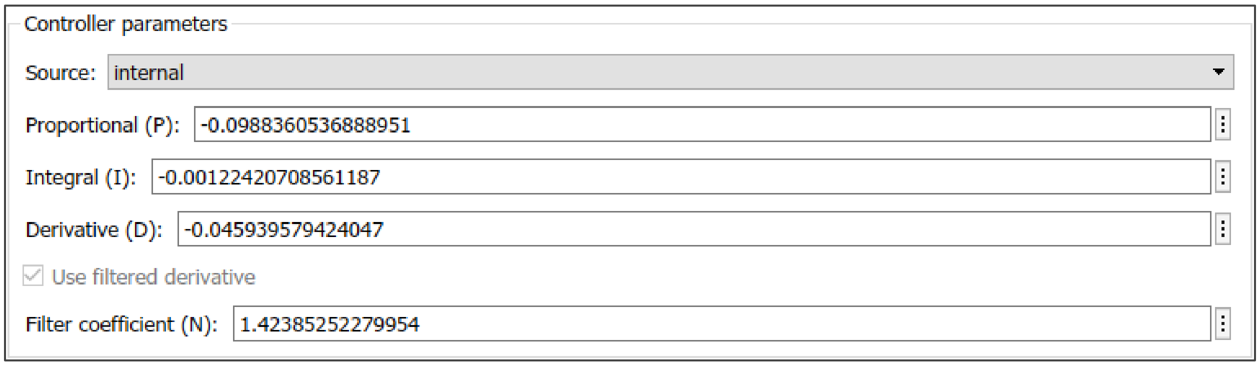
Proportional (P): 1821.17179797099

Integral (I): 28963.2055527084

Derivative (D): 28.119962782906

☑ Use filtered derivative

Filter coefficient (N): 1850.75173770887


Controller parameters

Source: internal

Proportional (P): -0.0988360536888951

Integral (I): -0.00122420708561187

Derivative (D): -0.045939579424047

☑ Use filtered derivative

Filter coefficient (N): 1.42385252279954

Figure 9 - Angle PID (top) & Position PID (bottom)

Simulation

The simulation is run in MATLAB and is visualized as the 3D model of the robot. Before simulating control, the uncontrolled robot is simulated to verify that everything is working as expected. After everything has been verified to work correctly, the PID controllers are activated and the simulation is run again. A disturbance block is added to simulate the robot being pushed.

The simulation results are recorded as an .mp4 file.

Simulation results

The simulation results can be view on YouTube at the following link:

https://youtu.be/9-L7qeeoDtc

or from the Github project page.

https://github.com/NikolaTotev/Teensy-Balance-Bot-Mk_1

The simulations show that without control the robot falls over and that when the PID controller is activated, the robot stays upright and keeps a position of 0. Any disturbances are also corrected.

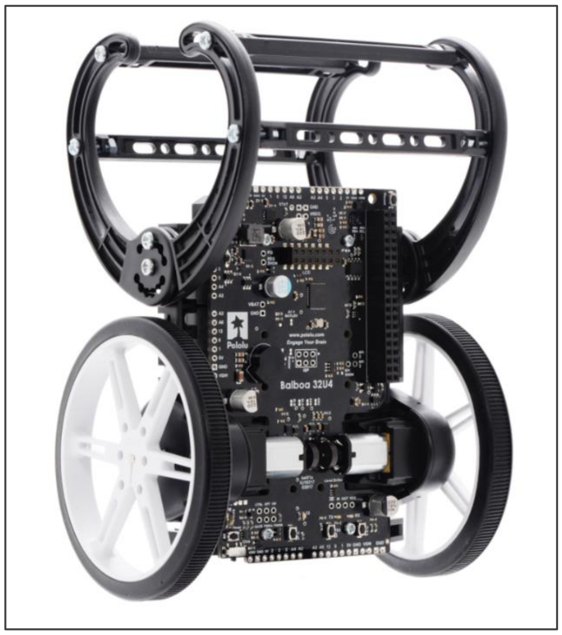
Comparison with existing solutions

This section analyzes current solutions on the market that have similar functionality. This comparison will consider the functionality as well as the use cases of the products. This is because there are products/projects that have related functionality, but they are used in industrial applications or are much larger or both.

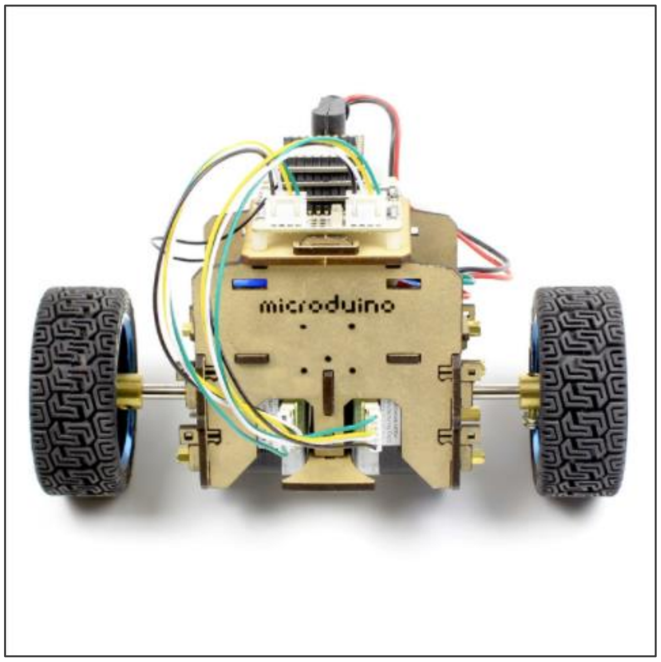
Existing solutions

Due to the limited number of products that have both similar functionality and use cases, only two solutions will be compared.

- **Balboa 32U4 Balancing Robot by** Pololu. This is a kit that can be assembled, wheels and motors need to be bought separately. During the rest of the comparison this will be referred to as ***Balboa*** for short. *(Photo from* (Pololu, 2021)*)*



- **Microduino Self-Balancing Robot by** Microduino. This is also a kit and comes with everything required. During the rest of the comparison this will be referred to as ***Mircoduino*** for short. *(Photo from* (Robotshop, 2021)*)*



Features

This section will look at the features each of the existing solutions have, and then they will be compared to the Teensy BalanceBot Mk1 based on a set of criteria that will be discussed in a following subsection. There is no mention of implemented functions, as both are kits and have to be programed by the end user.

Balboa

- Size - 118 × 112 × 80 mm
- Weight - 200g
- Microcontroller - Arduino-compatible ATmega32U4
- Single control board
- Integrated IMU
- Integrated motor drivers
- Integrated quadrature encoders
- Interface to Raspberry Pi
- Requires brushshed motors
- 6 AA Batteries

Microduino

- Size 19.5x8.5x11cm
- Weight – 650 g
- Control via Bluetooth
- Microcontroller Atmel ATmega1284P/ATmega644PA
- Long-distance Communication Module: Microduino-nRF24
- Stepper motors
- Two 3.7V Li-ion batteries

Comparison Criteria

As the list shows, each robot has similar features, and some that are present in one but not the other. To create a fairer comparison all three robots will be compared with the following criteria:

- Microcontroller model
- IMU & IMU Model
- Motor type
- Motor driver
- Encoders
- Battery Type

Product	MCU Model	IMU & IMU Model	Motors	Motor Driver	Has Encoders	Battery Type
Balboa	ATmega32U4	Yes - unknown model	Brushed - 30:1 Micro Metal Gearmotor HPCB 6V	Integrated- Unknown model	Yes	AA - Rechargeable
Microduino	ATmega1284P ATmega644P	Yes – Unknown model	Stepper – Unknown NEMA Model	Integrated – Unknown model	No	3.7V – Li-On
Teensy BalanceBot Mk1	Teensy 4.1 - ARM Cortex-M7	Yes – Pololu LSM6DS33	Brushed – Generic geared motors	Discrete – Cytron MDD3A	Yes	3.7V – Li-On

Software Development tools

This section will briefly discuss the choice of software development tools as it is considered an important part of the project development.

The primary development software is Visual Studio with the Visual Micro add-on. This software was chosen over others such as Visual Studio Code and the Platform IO plugin or the Arduino IDE, because it offered more stability during development, as well as more features that aid in the development process.

Development

This section describes the complete development process in detail. It is divided into two sections, hardware, and software. Each section contains all the details that were taken into account when developing the system, everything from microcontroller, sensor and actuator selection to material choices, assembly information and software development.

Hardware

This section will discuss in detail the hardware development process of BalanceBot Mk1.

Microcontroller

The microcontroller of choice for the Teensy BalanceBot Mk1 as the as the name suggests is the Teensy 4.1.

The Teensy was chosen as the MCU for this project, because of the following characteristics:

- It has a small formfactor and does not require a lot of space. However, despite the small form factor it still has a lot of pins and interfaces that can be used for any future additions such as lights, screens, actuators and sensors.
- 32-bit processor - The processor on the Teensy is a 32-bit Arm Cortex M7, it runs at 600Mhz and has a lot of functionality, such as support for floating point operations.
- Good support – the company behind the Teensy, pjrc has very good support for their products.

Sensors

This section discusses what kind of sensors are used and why.

Overview & Considerations

Due to the basic functionality required, the BalanceBot Mk1 needs only 2 sensors – and IMU and encoders for the wheels. Due to limited resources available for the development of the BalanceBot, the only available sensors are hobby grade ones. They are inexpensive and offer acceptable functionality.

IMU

The IMU being used in the BalanceBot is the LSM6DS33 breakout board from Pololu. This IMU module has a gyroscope and accelerometer, which means it can be used together with a complementary filter for sensor fusion.

It can use I2C or SPI, in the case of this project I2C is used. The sensor has a 16bit reading per axis for both the gyro and accelerometer. It provides good accuracy for the price and has a small form factor that allows it to fit almost anywhere.

Encoders

The Teensy BalanceBot Mk1 uses the magnetic encoder pair kit from pololu. They are meant to be used on DC motors with an extended shaft, however they were they only encoders available at the time of development. To use them correctly, a special motor housing was 3D modelled and printed to mount the PCB of the encoder and a special shaft adapter was also 3D printed to mount to the output shaft of the geared DC motor being used. More information regarding the design and creation of the 3D printed parts can be found in the “CAD Design” section. The issues that come from using the encoders this way are discussed in greater detail in the section “Future Development”.

Actuators

Actuators are essential for robots, and choosing the right actuator based on the use case is important. This section will discuss the actuator options for the BalanceBot Mk1, how they compare to one another and why one was chosen over the other.

Overview & Considerations

Since the BalanceBot Mk1 is a mobile robot, the ability to accurately control the rotation of the wheels is important. The motors also need to have enough power to control the robot, in the case of the Teensy BalanceBot Mk1, this means that if DC motors are used, they must be geared, as small DC motors do not have a lot of power. Another important thing to consider is being able to read the actual position of the motor versus the commanded position. This is done through encoders, and as mentioned before in the sensors section, the BalanceBot Mk1, uses magnetic encoders, this adds further limitations to the choice of motor.

Motors

When it comes to motors, as mentioned, one option is to use geared DC motors, the other is to use stepper motors. For this project, DC motors are used, this is because mounting an encoder to a DC motor is much easier. Another reason for using DC motors is power, stepper motors are precise, but do not usually come with any kind of gears. The stepper motors available at the time of development, were not as strong as the DC motors and therefore despite the DC motors being more inaccurate, they were chosen as actuator for the BalanceBot Mk1. Issues, possible solutions and future upgrades regarding the actuators are discussed in the “Future Development” section of this document.

Motor Driver

The motor driving being used in the BalanceBot Mk1 is the Cytron MDD3A driver. It has a maximum rating of 4-16V, 3A and can control 2 DC motors at once or 1 stepper motor. The input is PWM for speed and a high or low logic level for direction control. The motor driver has an integrated voltage regulator that supplies 5V.

CAD Design & Manufacturing

This section examines the main structure of the robot, how it was designed, what are the main sections of the robot, material choices and manufacturing methods.

CAD Design

Since the BalanceBot Mk1 has a small number of parts, in this section they will be discussed in some detail.

Battery compartment

The battery compartment is located at the bottom part of the robot, it houses 2 Panasonic NCR18650PF Li-On rechargeable batteries. The batteries are held in place by TPU springs with copper tape on the surface for conducting electricity. The electrical connections can be seen in the “Electronics” section.

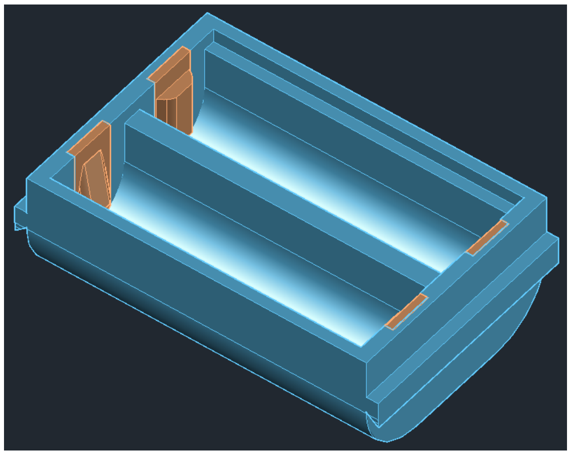


Figure 10 - Battery Compartment

Wheels

The wheels have a diameter of 100mm. They are comprised of two parts – a solid plastic hub that attaches with frictions and a keyed hole to the motor gearbox and a soft TPU tire that locks onto the hub with notches.

Custom wheels were designed because the ones that came with the motors were too small. Thanks to the soft TPU and sturdy hub, these custom wheels perform as good as the default ones.

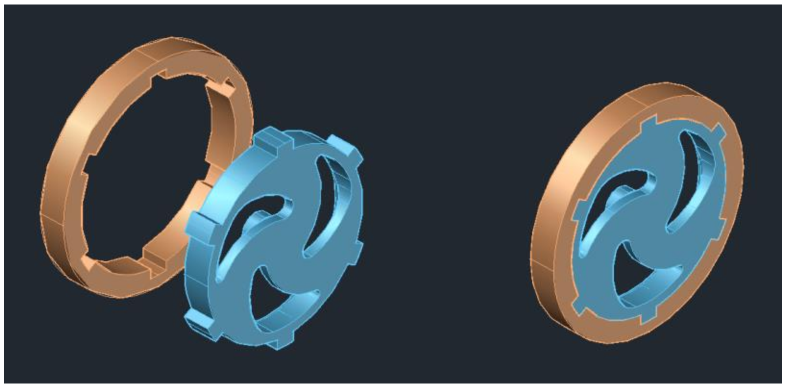


Figure 11 – Wheels & tires

Encoder mount

The encoder mounts server two functions, they hold the encoder PCB at the right distance from the motor and they help to hold the motor secure. The encoder PCB is soldered onto another PCB, that slides into the slot of the mount. Below the encoder, there is a hole for the wires. There is no need for securing the PCB as it is a friction fit and because the robot never goes inverted.

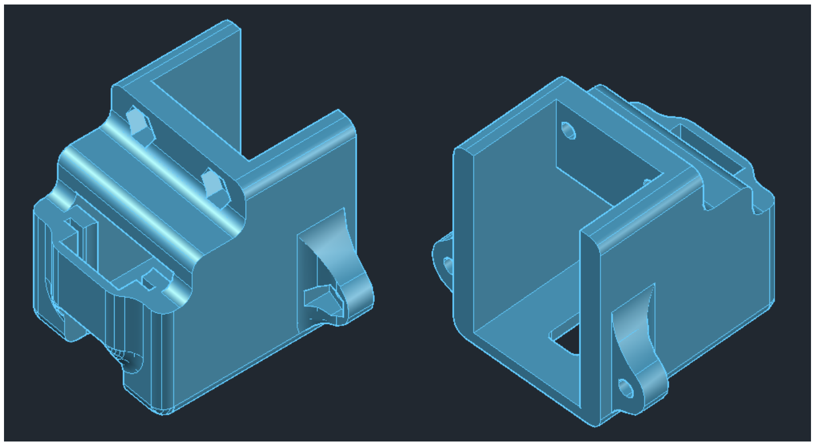


Figure 12 - Encoder mount

Bottom side panels

The bottom side panels hold the motors and encoder mounts. They also have dovetail rails, that are used for mounting the battery compartment. At the top of the bottom panels, a bracket attaches that is used for connecting the middle panel.

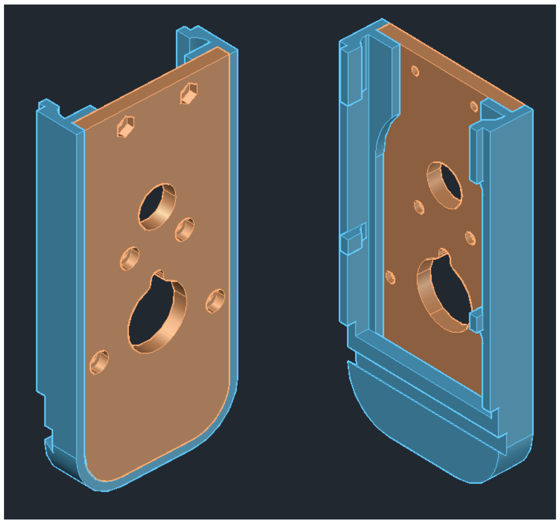


Figure 13 - Bottom side panels

Middle panel

The middle panel is the interface between the bottom half and the top half. It has holes for cable management and room for a small breadboard that holds the IMU. There are two sets of mounting holes – one set for the bottom mounting holes and another for the vertical brackets that are used for mounting the top panels.

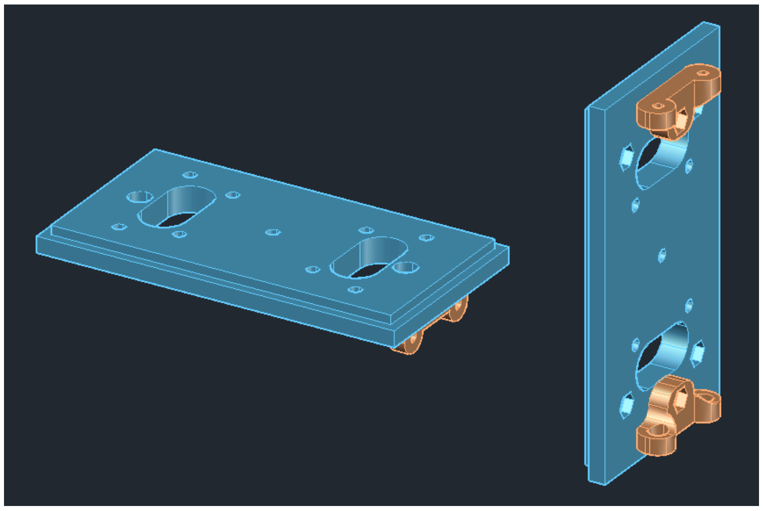


Figure 14 - Middle panel with bottom brackets

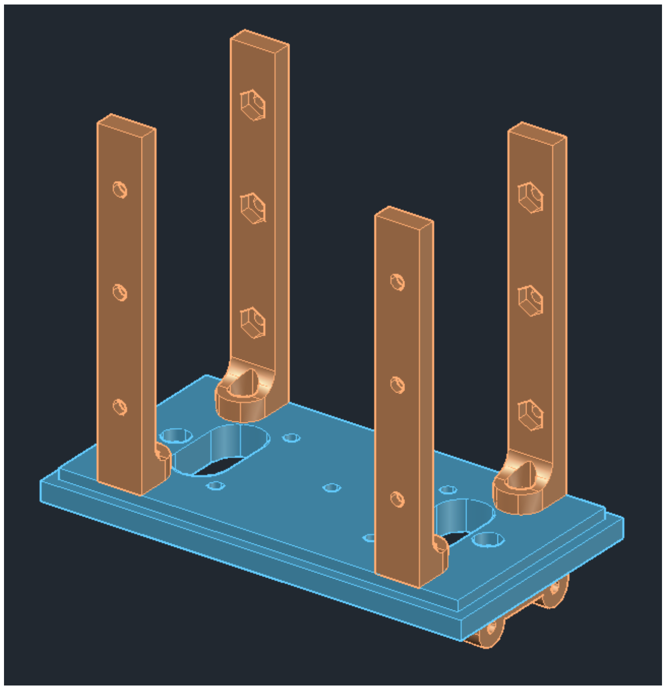
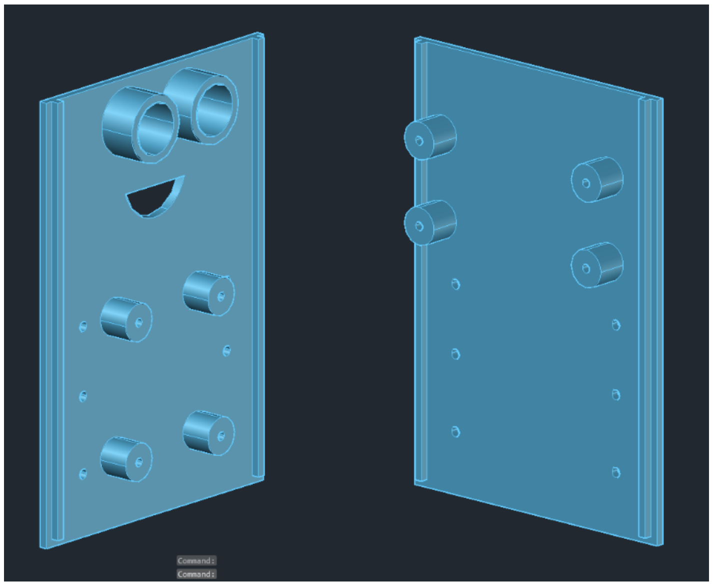


Figure 15 - Vertical brackets

Top panels (front and back)

The top panels create the electronics bay that contains the MCU and motor driver. The there are holes for mounting PCB standoffs to which the components are mounted to with small M3 screws. The top panels as mentioned before are attached to the middle plate with vertical brackets (two for each panel) and six screws are used for securing the panels to the brackets.



```
Command:
Command:
```

Figure 16 - Front and back top panels

Top Cover

The top cover is used for mounting master power switch and for adding additional weight at the top of the robot.

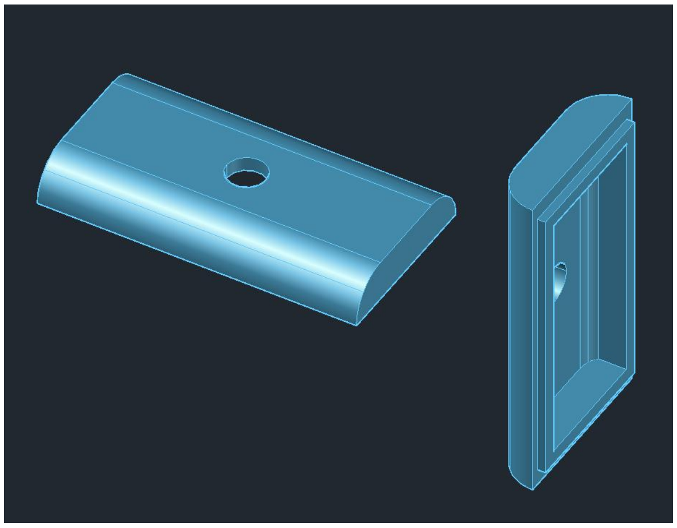


Figure 17 - Top cover

Manufacturing Method

The manufacturing method for the BalanceBot Mk1 is 3D printing. It allows for rapid prototyping. It is also cheap for allows for making more complex parts.

Material Choices

The materials used for the parts of the BalanceBot are PLA and TPU.

 PLA was chosen because it is easier to print than ABS. PLA is weaker than ABS, but this robot does not have any parts that experience significant forces

TPU with a hardness of 70A on the Shore scale. TPU is used mainly for the tires as they needed to be soft and provide grip. It is difficult to print, but with the right settings the results are good.

Electronics

This section contains all of the information about the electronics of the BalanceBot Mk1. Connections between individual components are discussed, what connectors and wires are chosen.

Circuit Diagrams

Figure 18 shows the circuit diagram of the system. A larger version of the schematic can be found on the Github (Nikola, 2021) page of the project.

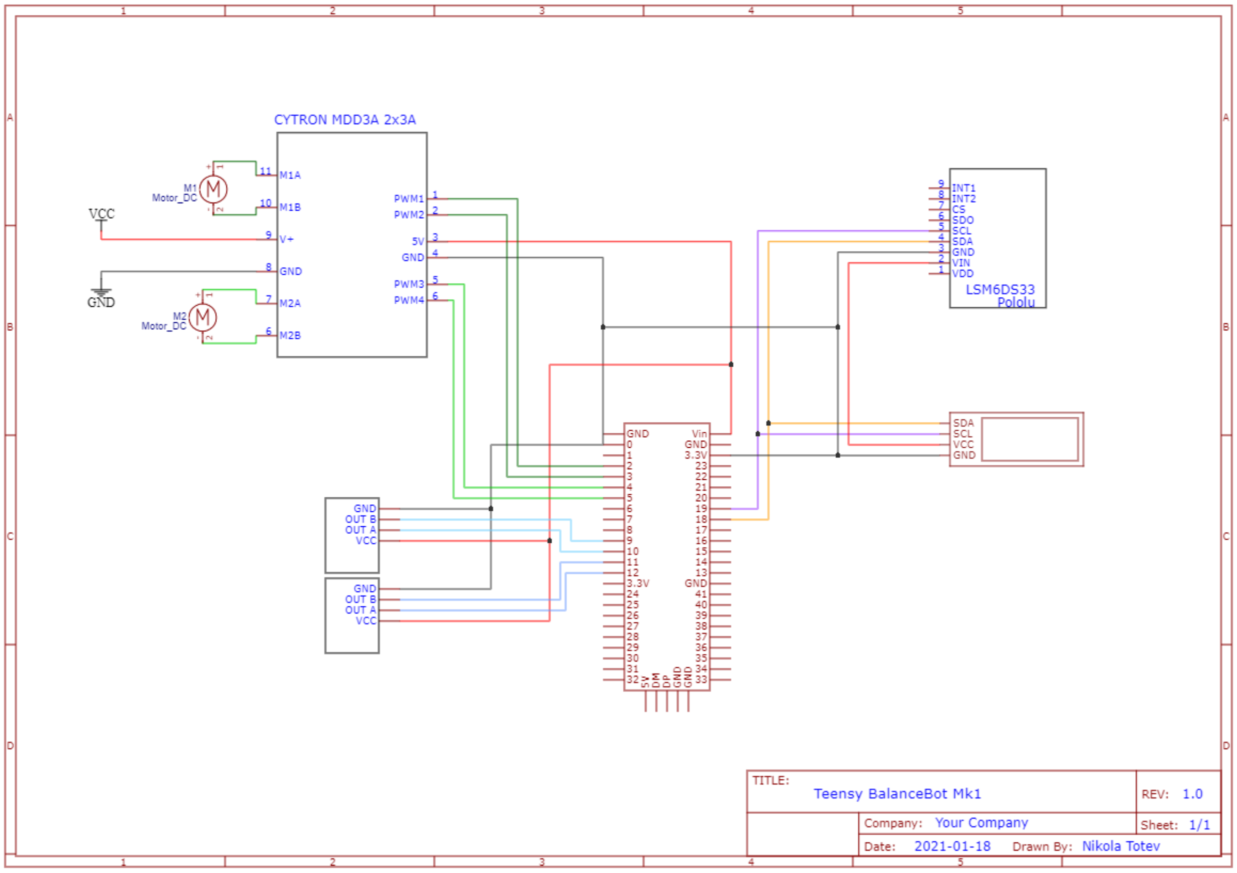


Figure 18 - System circuit diagram

Microcontroller Mount

The microcontroller has pin headers that need to be plugged into headers. To facilitate easy mounting a demounting of the MCU, as well as providing testing points for all of the pins, two double rows of headers were soldered onto a small 3x7cm prototype board. Female JST connecters were installed and wired to the correct pins to allow for easier connection of the other components of the robot.

Connectors & Wires

- **Connectors**

The connectors used in the BalanceBot Mk1 are 2 and 4 pin JST connectors and Dupont connectors. JST connectors are used for the encoders and IMU, and a male JST connector is used for one side of the wires going to the driver board. Dupont connectors are used to connect to the driver board.

JST connectors provide a robust connection between components, and unlike the Dupont connectors they do not easily come apart by themselves. The only reason for using Dupont connectors is because the driver board comes with them, and it was not practical to de-solder them.

Dupont connectors are also used for connecting to the breadboard that the IMU is mounted to. The connection method for the IMU has flaws, these issues and possible solutions are discussed in the "Future Development" section.

- **Wires**

The wires used are multistranded with either silicone or PVC insulation. The preferred insulation is silicone as it is very flexible and is easier to work with in tight spaces, however PVC needed to be used as it was the insulation that the Dupont connector cables came with.

During the development process, as expected the silicone wires were more durable and easier to handle.

Power

Batteries & Battery Holder

As mentioned before the batter holder is 3d printed. The primary material is PLA and the springs that ensure the batteries are held firmly in place is made of flexible TPU. The batteries are wired in series. The battery holder has a male JST connector that hooks up to the wiring harness that is connected to the power switch and the motor driver.

The batteries as mentioned before are 2 NCR18650PF 3.7V Li-On batteries, each with a capacity of 2700 mAh.

Power requirements

The power required for normal operation is around 0.8-0.9A at maximum draw.

Each motor draws up to 350mA.

The Teensy requires 100mA and is powered by 5V that is provided by the integrated power regulator on the motor driver board.

Final Assembly

This section will detail the final assembly details. Fastener choices and assembly strategies are discussed, and the final robot assembly is shown.

Fastener Choices

There are many choices when it comes to fasteners. For the size of the BalanceBot 1 and taking into account the 3D printer tolerances, the best fastener choice are M3 nuts and bolts. An indent is made to lock the nut in place, and one is also made to make the head of the blot be flush with the printed part.

CA (cyrano acrylate) glue is used to glue together the side panel assembly, because it is comprised of a frame and faceplate.

Assembly Strategies

As mentioned in the previous section, nuts and bolts are used. The primary assembly strategy is to use a central piece (the middle plate) to which brackets are attached, that then attach to the top or bottom pieces. This can be seen in figure 19.

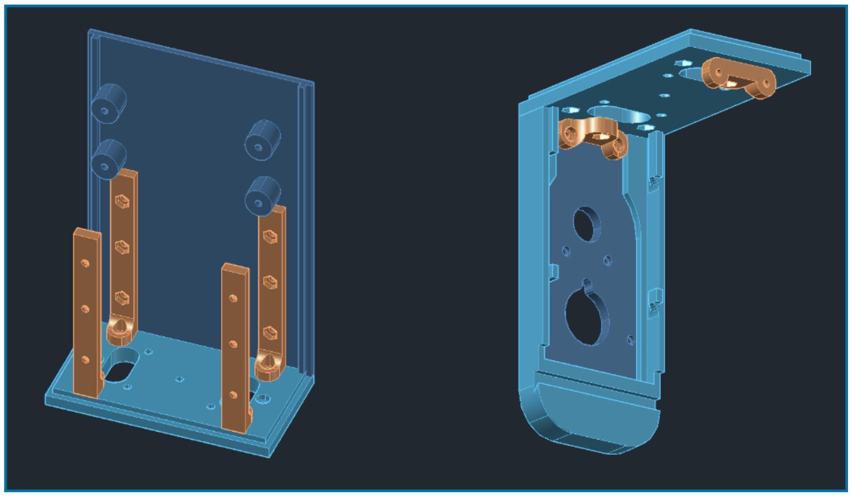


Figure 19 - Bracket assembly

The next strategy can be seen in figure 20, a small edge is made on the inside of the part, and on it rests a panel to create a solid face. When using such edges, CA glue is used to ensure everything is secure.

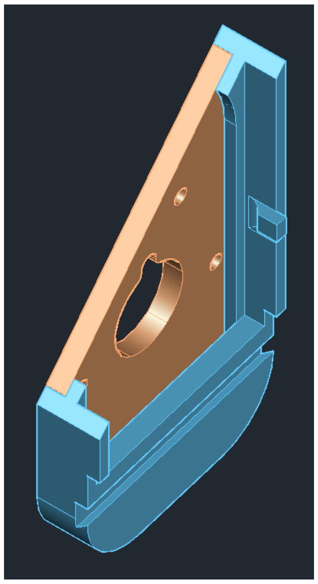


Figure 20 - Notch assembly

The last strategy can be seen in figure 21. The dovetail rail is an easy way to mount something that needs to stay in place but still easy to add and remove. In the case of the battery compartment, it is the perfect choice, because it allows for easy battery when it is time to recharge them.

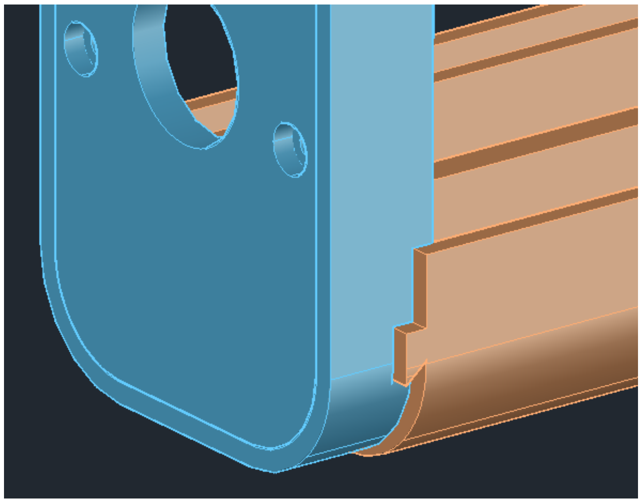


Figure 21 - Dovetail assembly

Assembly Steps

Step by step assembly instructions can be found at the project Github repo. The name of the file is AssemblyManual.pdf

Assembled Robot

Figure 22 shows the assembled robot.

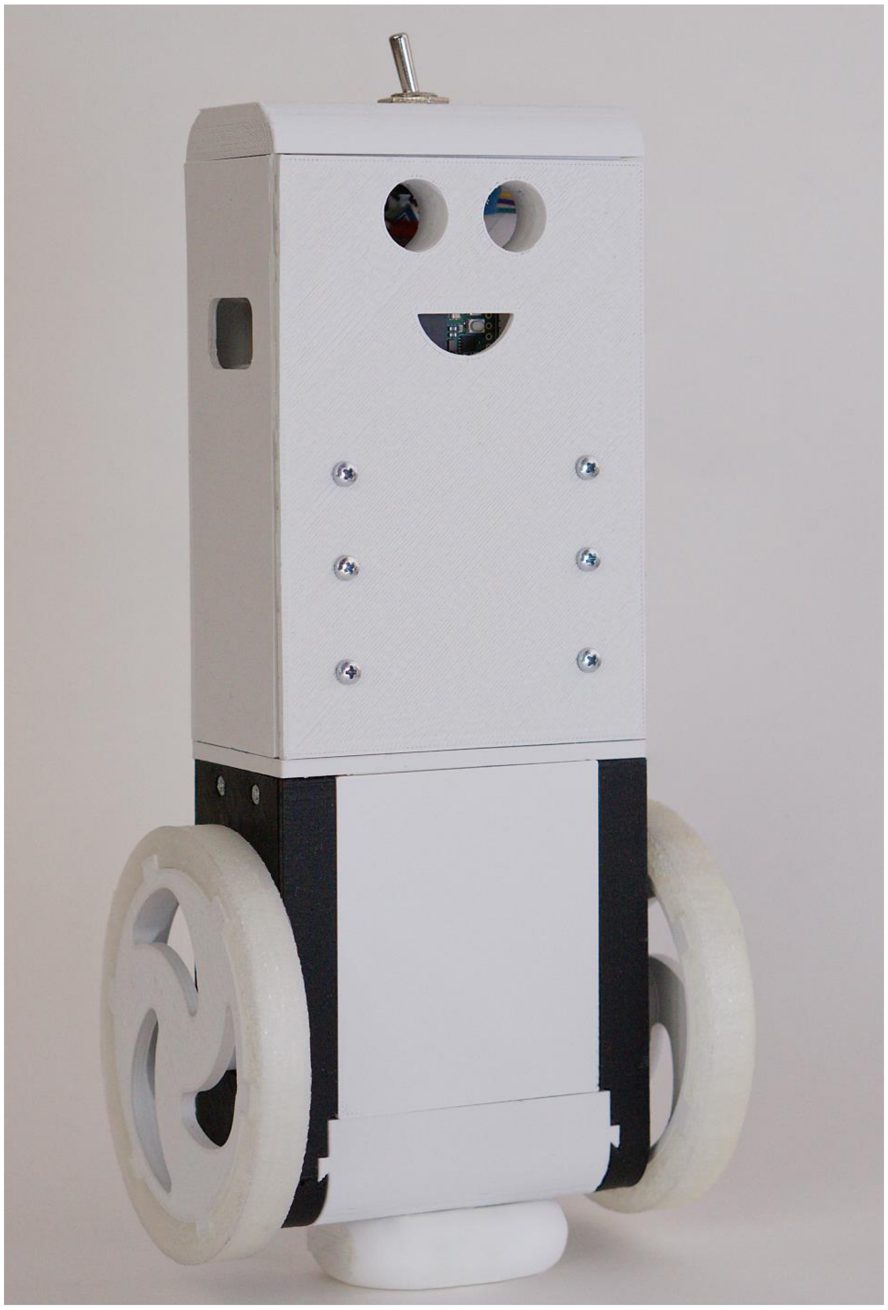


Figure 22 - Assembled robot

Software

This section will discuss in detail the software development process of BalanceBot Mk1.

Development strategy

The software development strategy used for the development of BalanceBot Mk1 was to design the different software components separately, test them to verify that they work and after that to combine all the working components. The software components are discussed in a subsequent section.

Software architecture

Software modules

- **IMU –** The IMU class handles all communication with the IMU. It provides functions that output DPS and acceleration for all the axes.
- **Encoder –** The encoder class contains the code required to read values from the encoder. It provides access to the number of ticks for each encoder as well as the last know direction.
- **Position Measurement –** The Position class handles the position calculation based on information from the encoders. It provides methods that return the current position for each wheel and an update method that must be called by the main loop of the program.
- **Angle Measurement –** The angle measurement class handles calculating and storing the current angle. It provides functions that return the current angle for the X and Y axis. This class must also be updated in the main loop of the program.
- **I2C Utilities –** i2c_utils class provides a wrapper for the functions from Wire.h. This wrapper is designed for easier reading from and writing to registers.
- **Motor driver –** This is class interfaces with the motor driver. It only provides one function – UpdateSpeed() and this function takes as arguments the speed and direction for the motor.

Final Structure

Figure 23 shows the final software structure. A larger version of the schematic can be found on the Github (Nikola, 2021) page of the project.

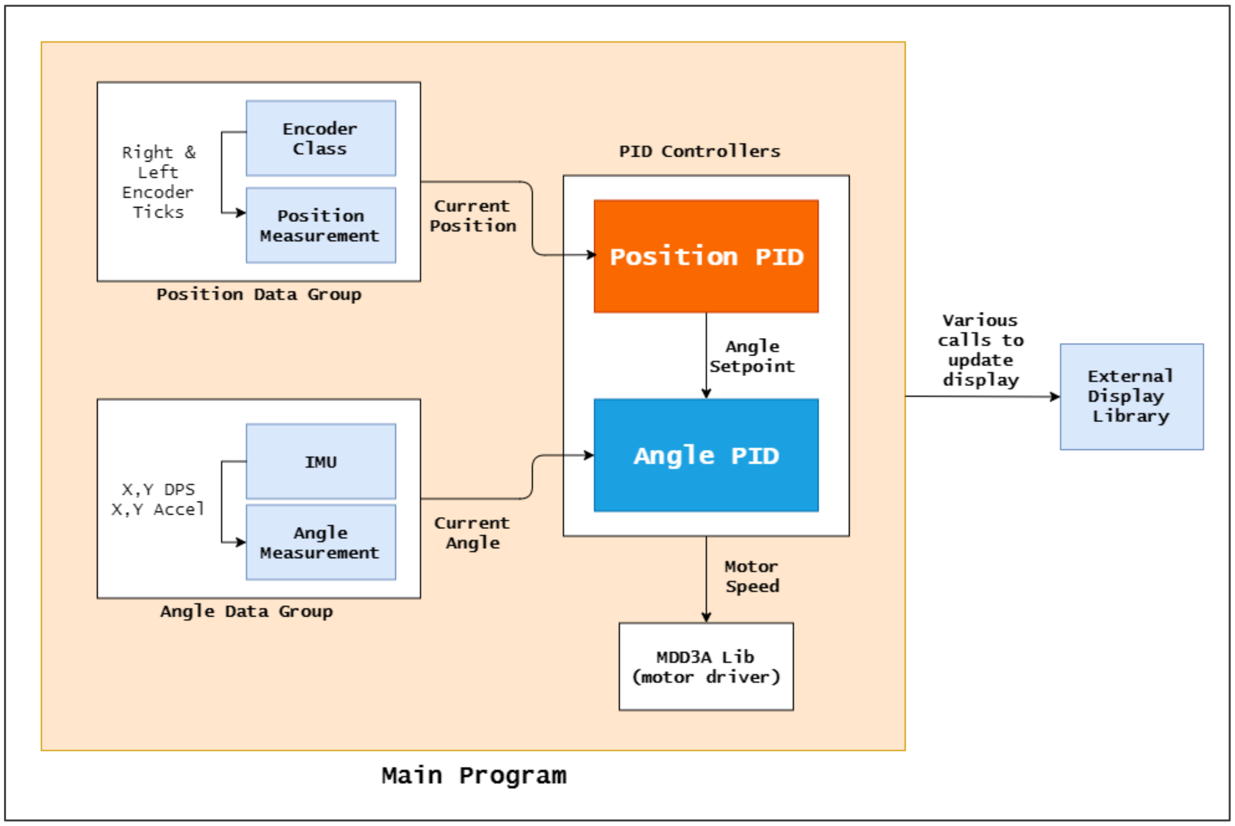


Figure 23 - Software Structure

Controller

PID Tuning

As mentioned in the “Automatic PID Tuning in MATLAB” section, MATLAB has an automatic tuning function, however it cannot be used for tuning the implemented PID controller. Because of this limitation, the PID tuning was done empirically. Once an acceptable performance was reached the PID was considered tuned.

Implemented Functionality

Balance mode

Balance mode is successfully implemented. The system does “shiver” a bit, but that does not affect the balancing functionality. Additional PID tuning can help to reduce the shiver, but other fixes such as a soft IMU mount are required. These fixes and improvements are analyzed in the “Future Development” section.

Position hold

Position hold is also implemented, however due to poor sensor performance it is not as accurate as the balance mode. The “Future Development” section has more information about the issues, and how they may be fixed.

Testing

Testing is a key step of the development process and this section will discuss testing strategies, test equipment and test results.

Testing strategy

Since most testable features for the Teensy BalanceBot Mk1 are tied to code, all tests have a mix between hardware and software tests.

The testing strategy used is to develop a single functionality –for example reading the IMU data and then testing if the hardware works and if the software is correctly reading and processing the data.

Testing small features often ensures that once all the components come together, they will work as intended. Thanks to this testing strategy almost no issues were found when all the components were combined into one system.

Testing equipment

The testing equipment used an oscilloscope, a multimeter, a bench power supply, and a computer.

The oscilloscope was used to check the type of signal being transmitted as well as for verifying that the system is working with the correct timing.

The multimeter was used in continuity mode to verify all of the solder connections.

The computer was used for uploading code to the MCU and verifying signal readings.

Test descriptions

This section will give a more detailed overview of the tests that were conducted.

Circuit Test

Circuit tests encompass PCB tests and wiring harness tests. The only PCB test that is required is the test for the MCU mount.

The check is done with the multimeter in continuity mode. A lead is placed in the female header and another lead is attached to the end of the connection (for example from the PIN that handles encoder input to the pin of the encoder) by testing this way, the MCU mount is checked and at the same time the wiring harness for a given sensor/actuator is tested.

Encoder Test

Hardware test

The encoder test starts by connecting the wires attached to the encoder PBC to the oscilloscope and then powering it on. The power in this case comes from the motor driver which is powered by the benchtop power supply. The PCB is put into the encoder mount. The magnets are mounted to the motor and the motor is put into the encoder mount as well.

To test if the expected square waves are generated, the motor is turned on with the built-in buttons on the motor driver. If everything is working the encoder should produce the waveform seen on figure 24.

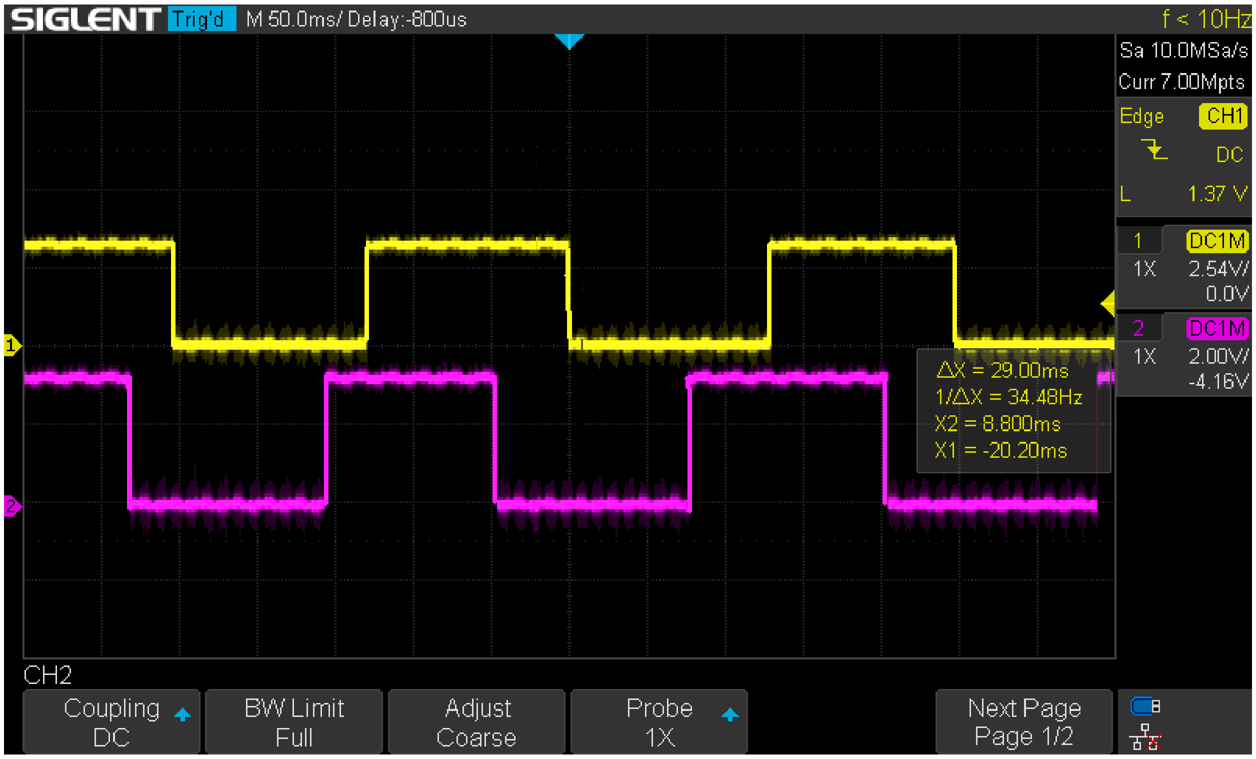


Figure 24 - Encoder waveform

Software test

The software test involves connecting the encoder to the MCU and then counting the ticks of the encoder with simple test code. Once this test code is verified it is improved and becomes the software module for working with the encoders.

Screen Test

The screen test involves attaching the pins of the OLED screen to the power and i2c pins of the MCU and then using the library from the manufacturer to test if the screen works.

IMU Test

The IMU test consists of attaching the power and i2c wires and then using test code to read basic DPS and acceleration data. After that the angle measurement code is written, where the gyroscope and accelerometer data are combined. With the angle measurement code written the results are tested by attaching the IMU to a breadboard and then tilting the breadboard to a known angle. If the angle on the screen is the same as the known angle, the test passes.

Motor & Motor Driver Test

The motors and motor driver are first tested by using the hardware buttons mounted to the driver board. Once its verified that the motors work and that the driver can control them, test code is written that controls the direction and speed. The output of that code is PWM wave and a high or low logic level. The output is check using an oscilloscope.

Once the output is checked, the MCU is connected to the motor driver and the code is run. The motors should cycle from minimum to maximum speed and then stop.

Future Development

This section discusses some options for future development, what issues the current project have and how they can be solved.

Current Issues

Encoder inaccuracy

Issue

At the moment, the way the encoders are mounted, one wheel revolution equals only 3 encoder ticks. This means that the positional accuracy is very bad and significant position drift can be observed before any kind of correction is applied.

Solution

There are several solutions, the first one would be to install better (meaning more steps per revolution) encoders to the wheels. Another one would be to mount the current encoders as the manufacturer intended. The encoders being used in the BalanceBot Mk1 are supposed to be used with DC motors that have an extended shaft at the back. As seen on figure 25 they are mounted to the rear of the motor and the magnets are mounted to the shaft.

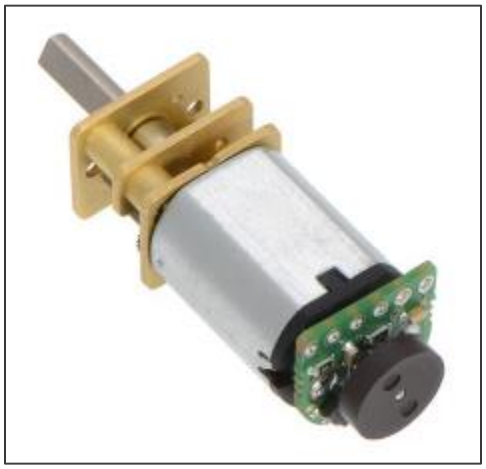


Figure 25 - Correct encoder mounting

IMU mounting

Issue

Right now the IMU is mounted to a breadboard. This mounting method creates a couple of issues. The first one is that all of the vibrations are picked up by the sensor and they create a noisy measurement which in turn creates a poor control response. Another issue is that the IMU is not perfectly level and can easily shift. This causes an inaccurate reading which introduces a lot of instability. The third issue is that the sensor is mounted right next to the motor driver board, since it is working with higher currents and because the motor direction changes rapidly, this can cause a significant amount of electromagnetic interference that again can cause a noisy sensor reading.

Solution

The solution to the first and second problem would be to create a custom PCB (either printed or from prototype board) to mount the IMU to. This board can then be attached to a 3D printed TPU frame/platform that absorbs some of the vibrations. The third problem can be solved by moving the IMU to a more suitable location.

Robot structure & assembly

Issue

The current structure is too small for any kind of improvements. Its also difficult to assemble and maintain.

The wheels are too thin and do not provide enough grip.

The side panels twist if enough force is applied.

It is difficult to remove the batteries from the battery compartment.

There is no wire management. If more sensors are added, there has to be some kind of wire management solution.

Solution

Unfortunately to solve the structural issues a complete redesign would be required. Along with solving the current issues a complete redesign would allow for correct mounting solutions for future sensor and actuator upgrades.

Motors

Issue

The motors being used are not of very good quality, the gearboxes have a lot of backlash and the wheel mounts are not level and some wheel wobble can be observed. There is also no way to mount encoders to these motors.

Solution

Metal gearboxes, such as the ones made by Pololu are a better option. Actobotics also make good motors that have integrated encoders. Another option might be to use stepper motors, but in that case a motor driver upgrade will be required, or at lease another motor driver needs to be purchased as the MDD3A can only control 1 stepper motor at a time.

Future Improvements

This section will explore some of the future improvements that can be made to the next iteration.

Hardware

LiDAR Sensor

A LiDAR sensor can be added for precise distance measurements. The LiDAR sensor can also be mounted on a rotating platform to provide a 360 field of "vison" that can be used for mapping the environment.

Sonar Sensor

A sonar sensor can be used for collision avoidance as it has a wider field of view. Even though it has a slightly less accurate distance measurement it can still be used together with the LiDAR module to map the environment.

Radio Control

Wireless modules, such as a Bluetooth or WiFi can be added to allow for remote control of the robot. A LoRa module can be added as well and it can be used for telemetry.

Functionality

Implementing GUI interface

A GUI interface can be created for a phone or computer. With it the robot can be controlled or reconfigured. A real-time video feed can also be added.

Adding direction control

Code can be added that takes in direction instructions and the robot can be made to go forward, backward and turn. This can be done from a remote control or the GUI interface.

Collision Avoidance

Collision avoidance can be added, this can work together with the direction control to help the operator avoid obstacles.

Environment mapping using ROS and SLAM

ROS(Robot Operating System) can be added to a raspberry pi on the robot. With it SLAM (Simultaneous Localization and Mapping) can be achieved. With this kind of functionality the robot can be made completely autonomous.

References

Nikola, T. (2021, 01 28). *Project Github Page*. Retrieved from Github: https://github.com/NikolaTotev/Teensy-Balance-Bot-Mk_1

Pololu. (2021, 1 18). *Pololu*. Retrieved from Pololu.com: https://www.pololu.com/product/3575

R.G. Lerner, G. T. (1991). *Encyclopaedia of Physics (second Edition)*. VHC Publishers.

Robotshop. (2021, 1 18). *Robotshop*. Retrieved from Robotshop: https://www.robotshop.com/en/microduino-self-balancing-robot-kit.html

Steven L. Brunton, J. N. (2017). *Data Driven Science & Engineering (Machine Learning, Dynamical Systems, and Control)*. Washington.