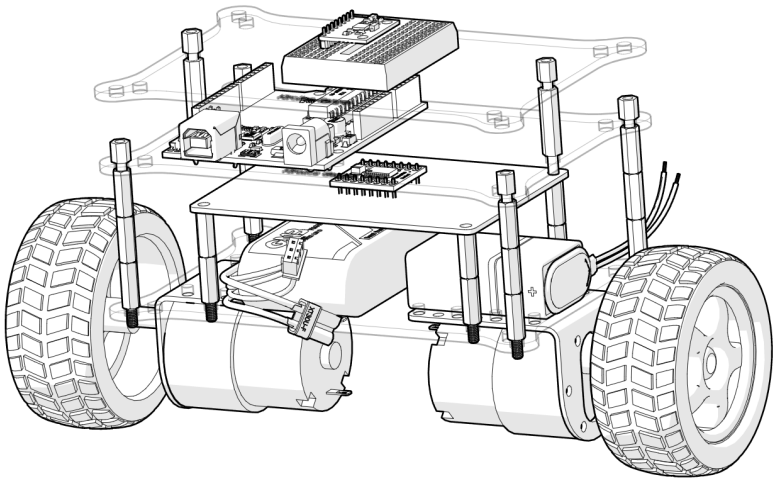
MCEN 5115 Mechatronics and Robotics

Building and Optimizing an Arduino-Controlled Inverted Pendulum Robot

Kirollos Gerges[1], Nicolas Garzione[1], and Nick Hudson[1]

[1]University of Colorado Boulder

December 10, 2021



Abstract

In the endeavour to explore the world of control systems, the team designed, built, and optimized a inverted pendulum robot that was able to successfully stand on its own. The robot utilized an MPU to read accelerometer and gyroscope data. Using this data, a custom PID controller was implemented to proportionally control the motors to maintain an upright position. The inverted pendulum challenge is a classic challenge in the world of control systems due to its inherently unstable nature. This challenge was met and the end result is a robot that can balance on its own.

1 Introduction

In the field of robotics, control is one of the three key elements for the necessary function of an autonomous system. Control is achieved through a controller, and the most common controller used is PID. PID stands for Proportional Integral Derivative and it is the controller used for this project. The premise of a self-balancing robot is an exercise in the design, implementation, and optimization of control systems. This was explored through the successful programming of this robot and resulted in it being able to stand on its own

2 Design

The two wheel robot was first designed in CAD for an overall picture of how the system would function. The CAD helped with initial evaluation and giving a rough idea of where the center of mass is. After a couple of iterations, the robot was adjusted for a very low center of mass. In the design process, the motor mounts were replaced to get the motors as close as possible to the lower "level" of the robot. All the heavy components were placed as low as possible on the robot to further help in lowering the center of mass.

Once constructed, the original motors were found to be insufficient to "catch" the robot as it was falling. They were replaced with higher power motors that can easily control the direction and orientation of the robot.

Components List:

1. MPU6050 (GY-521 module)
2. TB6612FNG Motor Controller
3. Arduino Uno

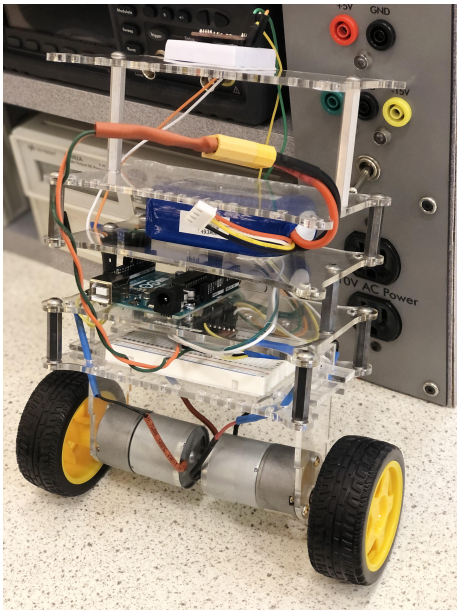


Figure 1: The original design of the robot, where the height and improper placement of components resulted in a high center of gravity.

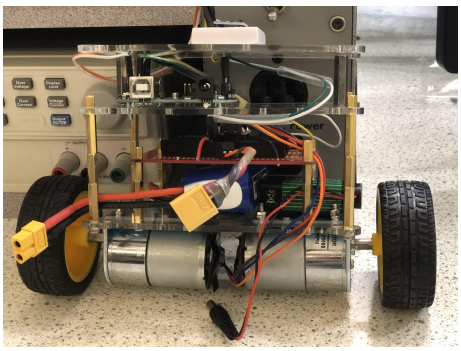


Figure 2: The robot after adjustments to the motors, height, and weight distribution of the robot were made.

4. LiPo Battery
5. 9V Battery
6. Motors

3 Theory

The PID controller is a three step process utilizing a position-time-acceleration approach to calculate a proportional response to an input. The value the controller begins with is the error, or the difference between the actual value and the value wanted for system stability. In the case of a self balancing robot, the system will be stable when the robot is upright, or at an angle of "0".

The proportional part of the controller is the value of gain added between the input error and

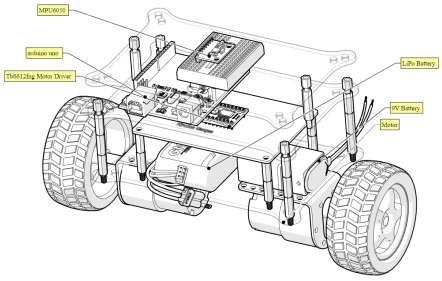


Figure 3: Components of the robot

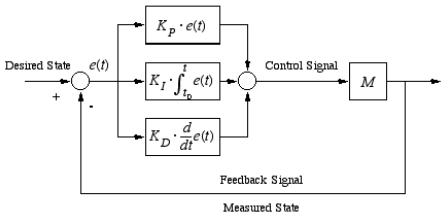


Figure 4: PID Controller Block Diagram

the motor response. More proportional value, more response by the motors per unit of error. The integral part of the controller is how fast the controller "winds up" to respond to error. Without an integral, with constant error the controller would not be able to over-compensate and realign back to center. In the case of this robot, that would look like the robot drifting one direction and the proportional response not being enough to correct for it and bring t back the other direction. The derivative part of the controller dampens the motion and prevents violent oscillation and over compensation in the system.

All together these three aspects of the controller allow for an appropriate response to the robot's current situation and allow the robot to remain at center.

In order to properly process MPU data, a filer is needed. If a filter was not included, the PID controller would react to noise in the system and throw the robot out of balance. The noise and error is generated from inherent flaws in accelerometers and gyroscopes. Gyroscopes tend to "drift" over time, producing inaccurate values when consistent values would be expected. Accelerometers are imprecise and are susceptible to outside forces. An example would be the force generated by the robot as it is correcting itself. This force can cause violent oscillations and over corrections in the robot, so it must b filtered out. A complimentary filter utilizes the best of an accelerometer and gyroscope combined to overcome their individual weaknesses.

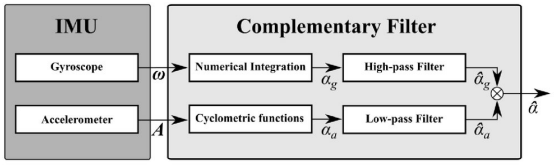


Figure 5: Complimentary Filter Diagram

4 Results

After extensive tuning, the PID values that resulted in the best performance were P:16 I:3 D:0.2. With these values, the robot could stand on its own. This was unexpected, since in referenced projects either the I or D values were much higher than the P value. However, in reference to the characteristics of this specific robot, it makes sense. The robot is squat, with its center of gravity very low It has has very high-powered motors that allow the robot to correct itself easily. This leads to less of a need for an integral value. With the addition of a complimentary filter, there is less of a need for damping since the data being read is "smoothed out". This reduces the need for a derivative value.

This tuning resulted in a successful implementation of the PID controller and the robot can balance on its own without intervention.

5 Conclusion and Future Improvements

Lowering the center of mass of the robot helped achieve an easier tuning of the PID controller. Along with this, the complementary filter added to the algorithm helped with decreasing the percentage of the overshoot in the robot response. In future endeavours, further calibrating the MPU along with precisely manufacturing the robot to have a more stable center of mass would result in better performance.

6 Acknowledgements

We would like to thank Dr. Reamon.

7 Appendix

Code Repository: github.com/nicolasgarzione/self-balance-robot
CAD: https://workbench.grabcad.com/workbench/projects/gcT19s8bA1KxCMZ2stMGVkM8Kg6wJEc_KGP404cn6gZTwF#/space/gck-CMbeClvbarTxgrQw5Q7tegI7ZlzhrYboEVhAYA4Okx