# Inverted Pendulum

## ME2240
## Project Report

Source: https://github.com/souryavarenya/Self-Balancing-bot

Video: https://www.youtube.com/watch?v=0pzGdVTDJ8c

**Mentored by**

**Prof. Sathyan S.**

Department of Mechanical Engineering

Indian Institute of Technology, Madras

**Team Members**

*Sourya Varenya, ME14B065*

*Sirish S., ME14B060*

*Ajay Rawat, ME14B100*

*Ashutosh Jha, ME14B148*

## Abstract

Everything around us involves physics and modelling the system is of a huge importance. Here, we model, analyse and construct a self-balancing robot which lets us understand the response of such system. This project revolves over the idea of an inverted pendulum, where a cart responds to the movement of the pendulum to compensate the fall of it. The most crucial part of this project is the tuning of the PID controller used to control the speed of the motors.

## System Modelling

The equations for x and θ were developed by drawing the free-body diagrams of the cart and the pendulum.

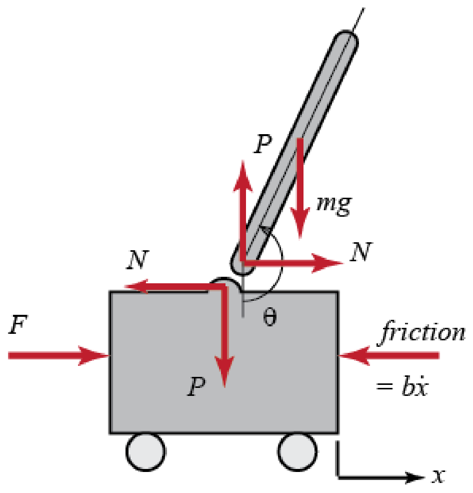


*Image 1: Free Body Diagram of the pendulum [1]*

The governing equations[2] of the physics are as follows:

$$\ddot{x} = \frac{1}{M}\sum\nolimits_{cart} F_x = \frac{1}{M}(F - N - b\dot{x})$$

$$\ddot{\theta} = \frac{1}{I}\sum\nolimits_{pend} \tau = \frac{1}{I}(-Nl\cos\theta - Pl\sin\theta)$$

Solving the equations for N and P, we get

$$N = m(\ddot{x} - l\dot{\theta}^2 sin\theta + l\ddot{\theta} cos\theta)$$
$$P = m\left(l\dot{\theta}^2 cos\theta + l\ddot{\theta} sin\theta\right) + mg$$

Similarly considering the F.B.D. of the wheels alone,

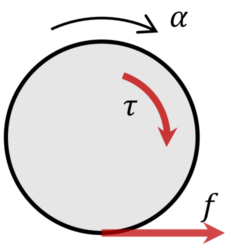


*Image 2: Free Body Diagram of the wheel*

$$F_{rest.} = 2\left(\tau - \frac{I_{wheel}\,\ddot{x}}{R}\right)$$

The total force on the cart is given by $F = F_{disturbance} + F_{restoring}$ where the input to the system is $F_{disturbance}$ which is set to an impulse.

## Simulations

Constructing a single transfer function for the whole system is one way of modelling but requires linearization since the system is non-linear. We are making use of Simulink which doesn't require linearization.

The upper half of the system model shows the PID control. The error from set angle (i.e. pi) is passed as a parameter to the PID block. The PID block in turn outputs a value from which when multiplied by a factor of the full scale torque (0 corresponds to 0 torque and 255 to full scale torque) is the output of the motor.

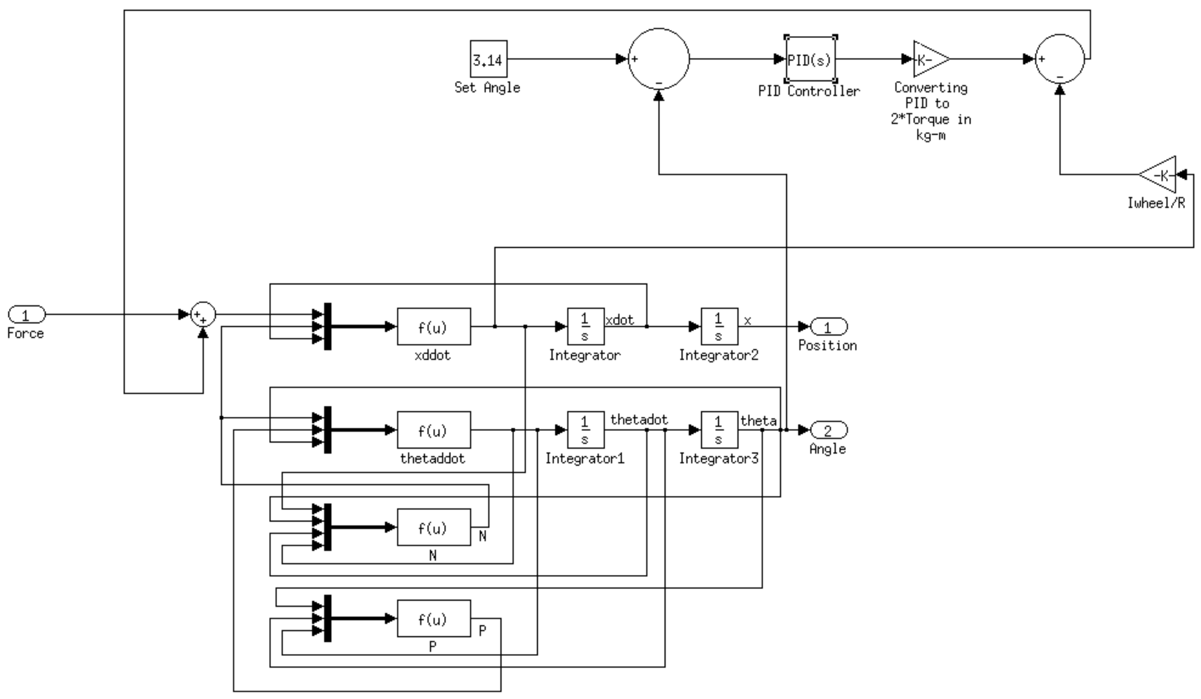


*Image 3: Simulink model of Inverted pendulum.*

The equations are then modelled subsequently in the $f(u)$ blocks and labelled. The integrator blocks integrate $\ddot{x}$ and $\ddot{\theta}$ to $x$ and $\theta$ respectively, which are our outputs and observed on the scope.

(We haven't observed x as the problem statement only wanted us to make sure the bot was stable. No conditions on the position of bot were imposed.)

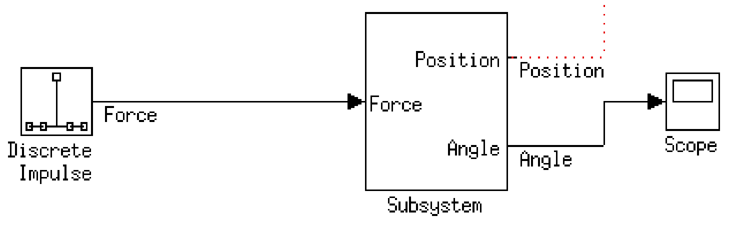
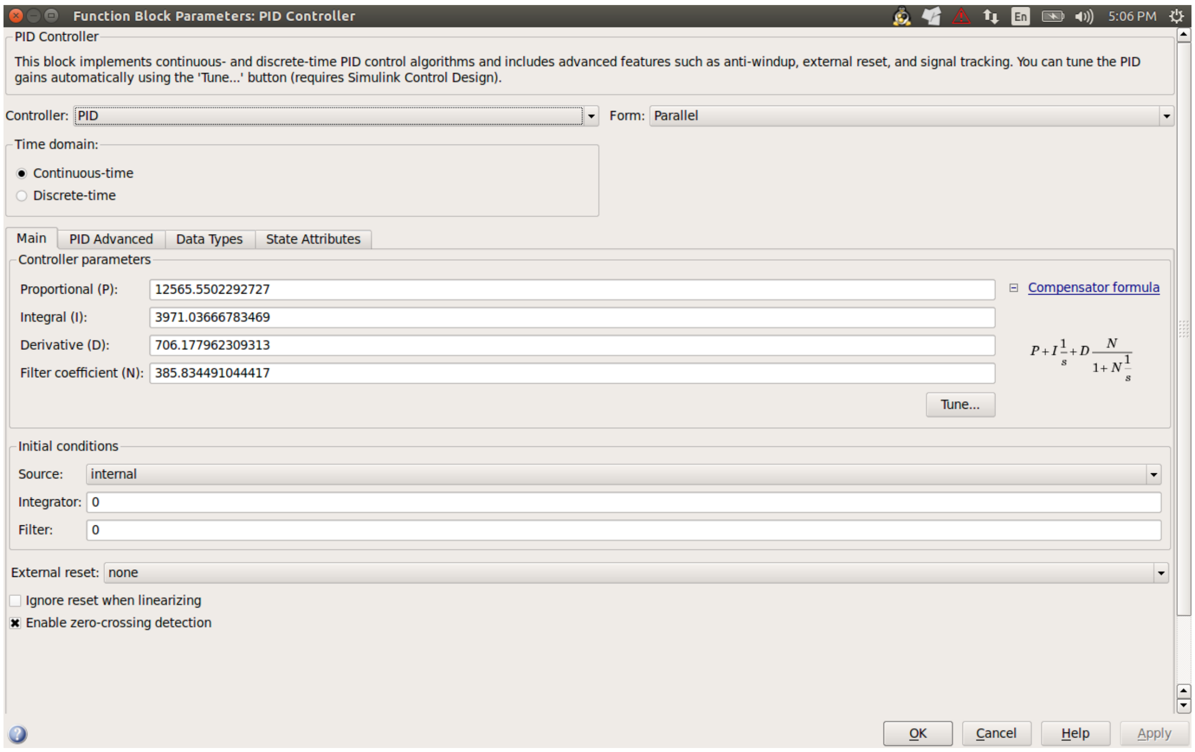


*Image 4: Overall Simulink model.*

We have chosen the PID controller. For finding $K_p$, $K_i$ and $K_d$, we make use of the auto-tune functionality.



Function Block Parameters: PID Controller

PID Controller

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

Main | PID Advanced | Data Types | State Attributes

Controller parameters

| | |
|---|---|
| Proportional (P): | 12565.5502292727 |
| Integral (I): | 3971.03666783469 |
| Derivative (D): | 706.177962309313 |
| Filter coefficient (N): | 385.834491044417 |

Compensator formula

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

Initial conditions

| | |
|---|---|
| Source: | internal |
| Integrator: | 0 |
| Filter: | 0 |

External reset: none

☐ Ignore reset when linearizing
☒ Enable zero-crossing detection

Image 5: PID Auto-tune Results

Due to the exclusion of transfer function of the motor and approximations, instead of choosing $K_p$, $K_i$ and $K_d$ values directly into the code, we chose to maintain the ratios of $K_p$, $K_i$ and $K_d$ instead.

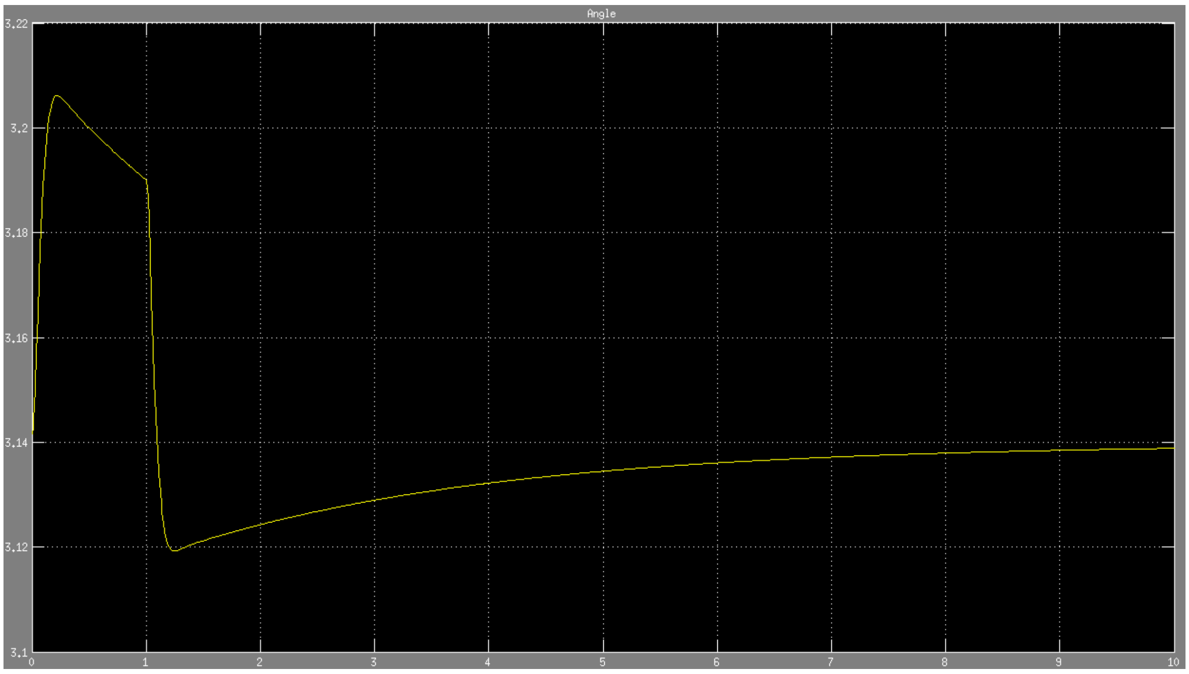


Image 6: System's Angle Response for an Impulse

## Chassis and Components

The Chassis basically consists of two plates, entirely built with plexiglass. As you can see in the *Image 7*, the two motors are mounted onto the chassis using L clamps. Two 9V radio batteries were used and place at a low level close to the motors as in the *Image 8*. All the wires from the motors were neatly insulated using heat shrinks.

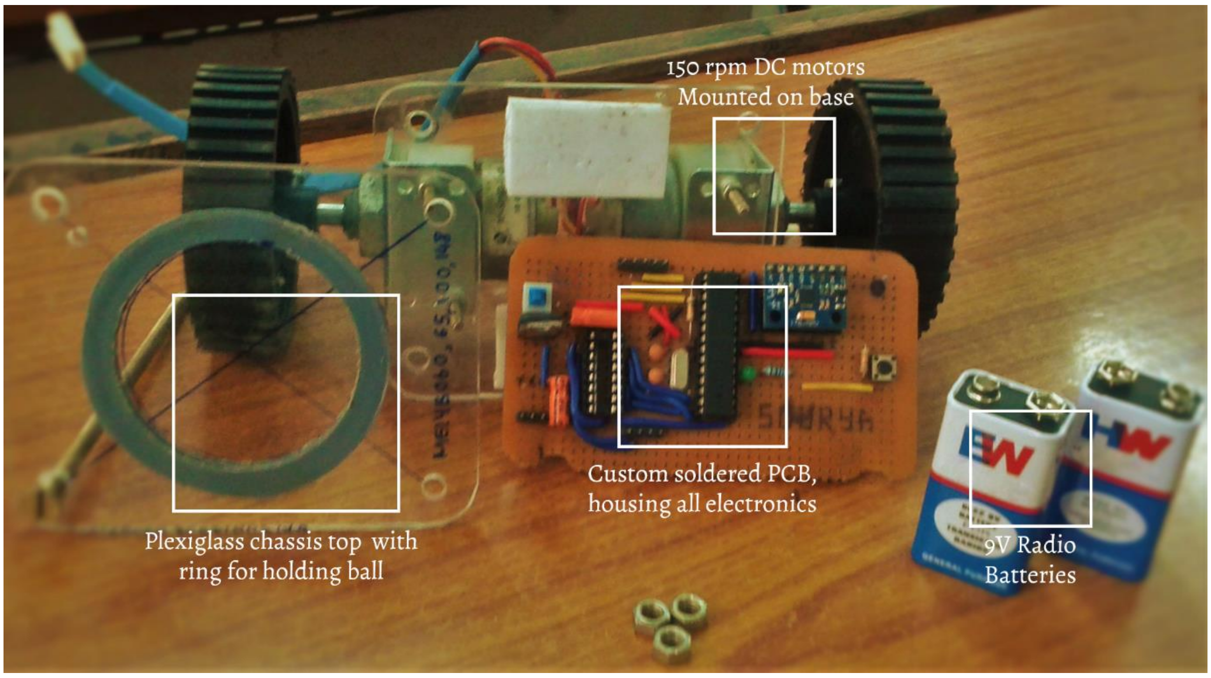


*Image 7: Overview of Components used*

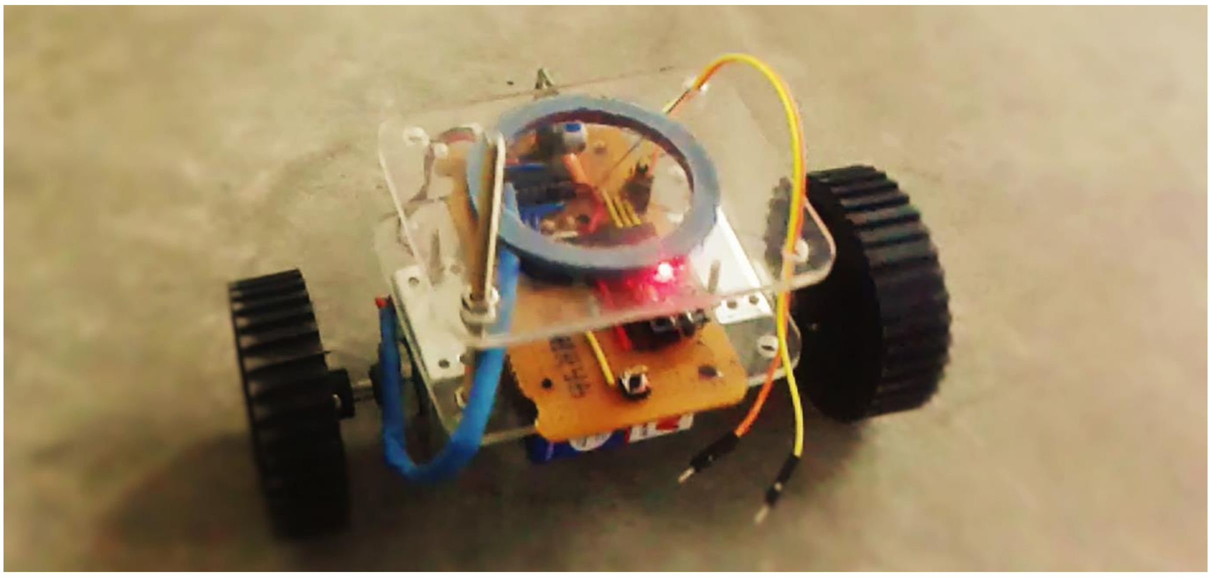


*Image 8: Self Balancing Robot in action*

## Electronics

The PCB underwent several iterations in design, size and placement of components. The ATmega 328P PU microcontroller was used instead of the entire Arduino. This massively reduced the clutter and was very compact and as small as 10cm×6cm

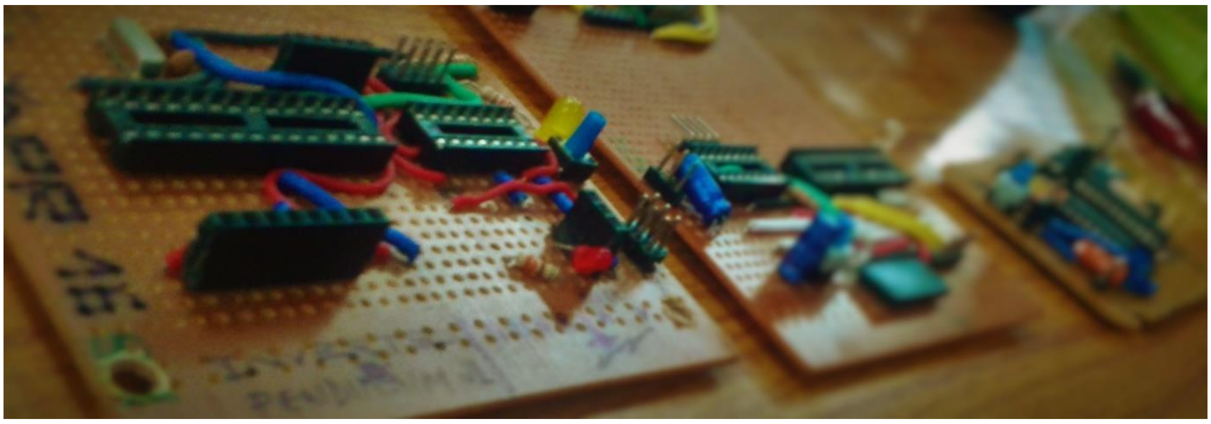


*Image 9: Three generations of boards, first and second failed*

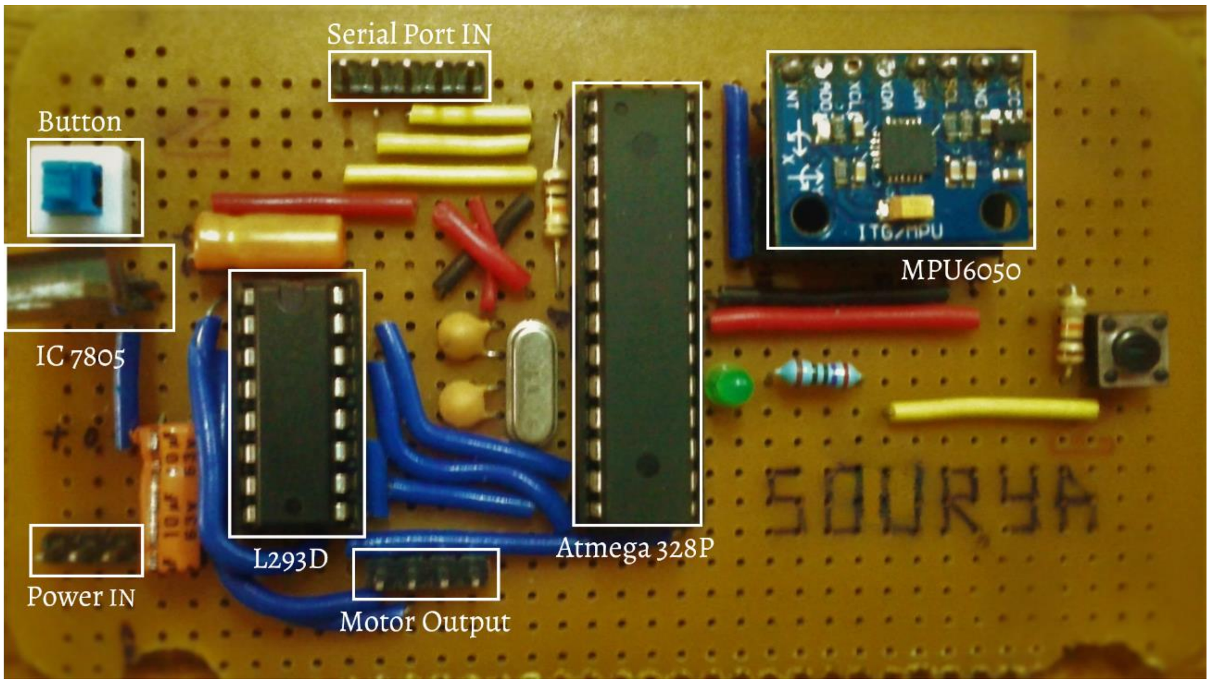


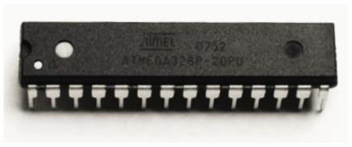
*Image 10: Design of Final PCB*

The PCB majorly housed the following components and their peripherals:

**MPU6050:** 6 axis inexpensive IMU with angular velocities and linear accelerations.

**L293D:** Motor Driver for scaling the 0-5V PWM output to 0-9V with sufficient currents

**ATmega 328P PU:** 8 Bit AVR Microcontroller from Atmel, running at 16.00 MHz, Programmed using Arduino IDE.



**IC 7805:** Voltage regulator for powering Arduino from 9V battery at 5V

## Code

In this section, we will take a look at how the code is structured. For obtaining the angle output from the MPU6050, an I$^2$C library is used. The raw values from the gyro sensor are prone to drift while those from the accelerometer are fluctuating. This code utilises a linearized, optimized complementary filter, which combines angle from accelerometer and gyroscope data to give accurate and fast angle output.

The angle input is fed to the PID controller as an error and $K_p$, $K_i$ and $K_d$ have been set to the values proportional to the simulation results. To stabilize a bit more, we have played around with the tuning parameters. There is still a scope to optimize the stability of the system.

The code can be found on this github repository:

https://github.com/souryavarenya/Self-Balancing-bot

## Conclusion

The transfer function model that has been used in the project shows satisfactory results in practice. The inverted pendulum is seen to respond well in the presence of external disturbances as well. We also conclude that the modelling of the equations in Simulink helps us to determine satisfactory ratios of values even in absence of information like the transfer function of the wheel.

All in all, the both the mechanical aspects such as the placement of centre of gravity and the electrical aspects work hand-in-hand to provide the stability to the inverted pendulum.

## Future Scope of the Project

There are many more exciting applications of this particular project:

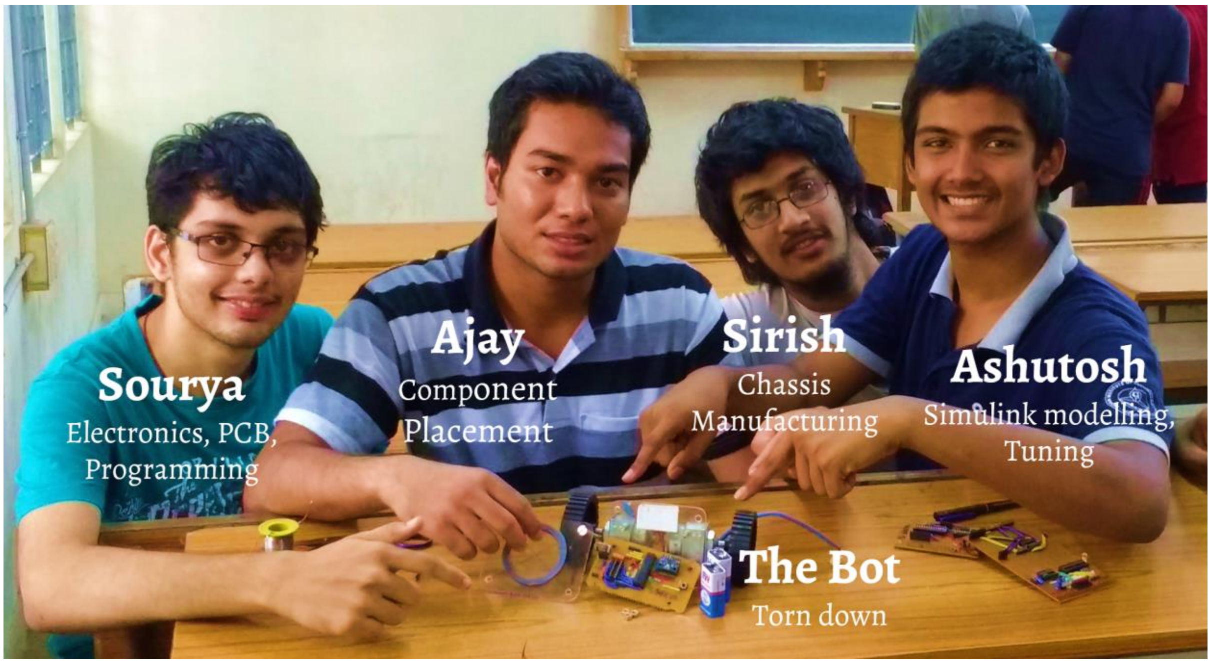
- We have used Transfer Function approach in our bot. There are few other approaches which we came across while researching on the theory such as State-Space Modelling, Polynomial functions which could possibly provide better stability.
- Another cool application of this is the stabilising of air-borne drones using quadrupeds. A TED talk for the same was shown in the class.

## References

*[1]*
*http://ctms.engin.umich.edu/CTMS/Content/InvertedPendulum/Simulink/Modeling/figures/pendulum2.png*

*[2]*
*http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SimulinkModeling*

## The Team


Sourya
Electronics, PCB, Programming
Ajay
Component Placement
Sirish
Chassis Manufacturing
Ashutosh
Simulink modelling, Tuning
The Bot
Torn down