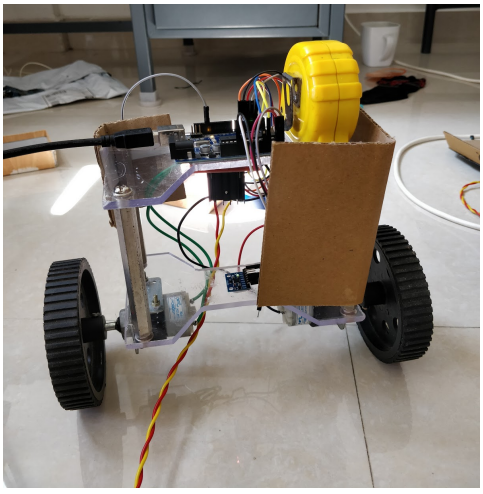
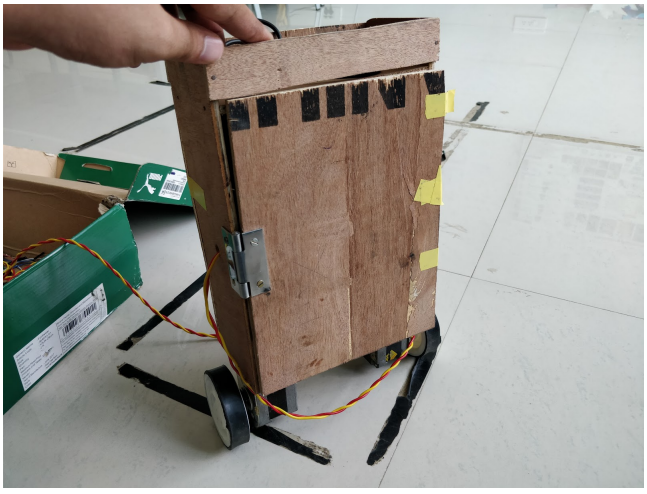
PRODUCT DEVELOPMENT PRACTICE

PROJECT REPORT

SELF BALANCING ROBOTIC PLATFORM





Rohan S (MDM15B017) **Govind KP (CED15I007)** **Prathmesh D(CED15I029)**

Himavanth R (ESD15I014) **Madhan (ESD15I016)** **Ajay Y (MDM15B022)**

CONTENTS

ABSTRACT

INTRODUCTION

STAGES OF PRODUCT DEVELOPMENT

1. VERSION 1
- Materials
- Construction
2. VERSION 2
- Materials
- Construction

WORKING OF PLATFORM

All Content related to this project can be found in this Drive folder- Self Balancing Robot Folder

ABSTRACT

A self-balancing robotic platform is not a new concept in the field of Robotics. Self-balancing robots are a topic of curiosity amongst students, roboticists, and hobbyists around the world. The fascinating aspect is the fact that it is a naturally unstable system which is being balanced for certain amount of time not only momentarily.

The project presents an attempt on developing an autonomous self-balancing robot. The project proposes a robotic vehicle that has an intelligence built in it such that it guides itself over two wheels with/without continuous human guidance. This robotic vehicle is controlled using Raspberry Pi 3 Model B+ (Broadcom BCM2837B0 processor). An inertial measurement unit (MPU 6050 IMU Sensor) is used to measure and report the robot's specific force and angular rate (motion tracking that combines a 3-axis gyroscope, 3-axis accelerometer, and a Digital Motion Processor). Depending on the input signal received, the processor redirects the robot to move in the same direction as the tilt i.e., towards forward direction if robot is falling forward and vice versa, by actuating the stepper motors (NEMA-14) interfaced to it through a L298N Motor Controller, thus achieving the task of balancing the platform on two wheels.

With the advent of Make in India, such products serving its core purpose are more than relevant and indispensable to the Indian subcontinent considering the present social context of automation in industry and elsewhere. This concept can be further developed to evolve it into a complete product capable of multi-tasking serving diverse functions, available at affordable prices hence sustaining itself in a competitive market.

INTRODUCTION:

A quick look at the range of mobile robots in existence reveals an enormous diversity in shape, form, and modes of mobility. However, one thing that most of them have in common is that they are *passively balanced* (i.e. their bodies are constantly in a state of stable equilibrium).

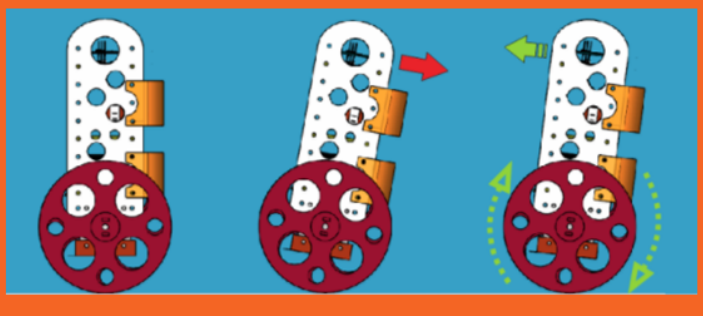
While this is perfectly logical in most cases, there are certain applications, such as Segways and humanoid robots, that take advantage of an unstable-equilibrium, inverted pendulum design to enhance their capabilities. While their self-balancing mechanisms may increase the complexity of their design, the benefits, which include greater maneuverability and stability, outweigh the costs.

Through this project we aim to look at the feasibility of application and ease of implementation of this concept to the field of autonomous trolleys and transport platforms.

PROBLEM TO TACKLE:

INVERTED PENDULUM

The physical problem that the robot solves is called the Inverted Pendulum. This is the same mechanism you need to balance an umbrella above your hand. The pivot point is under the center of mass of the object.



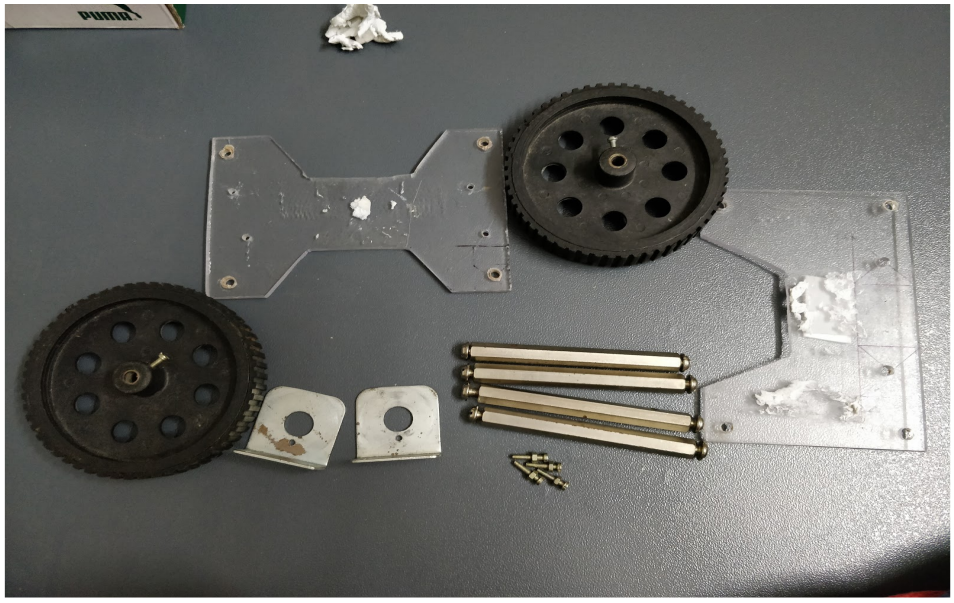
PRODUCT DEVELOPMENT STAGES:

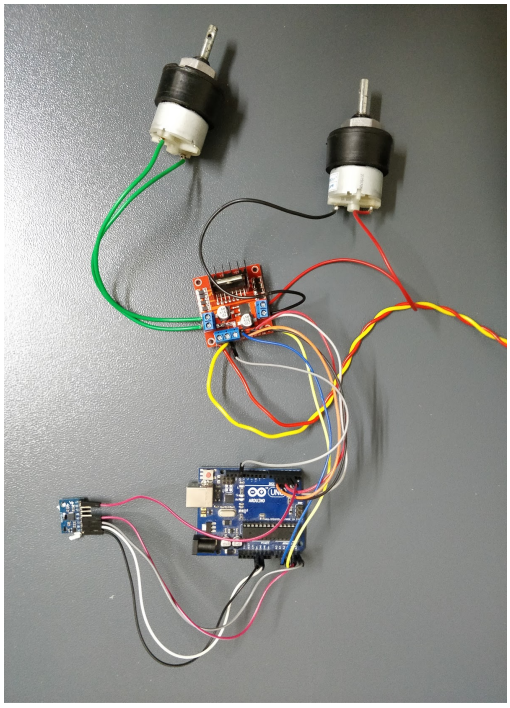
VERSION 1.0-

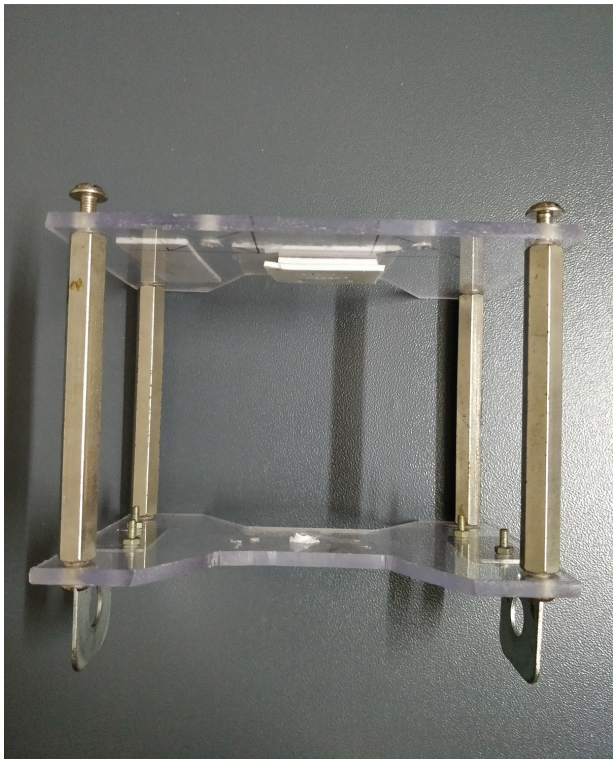
MATERIALS USED:

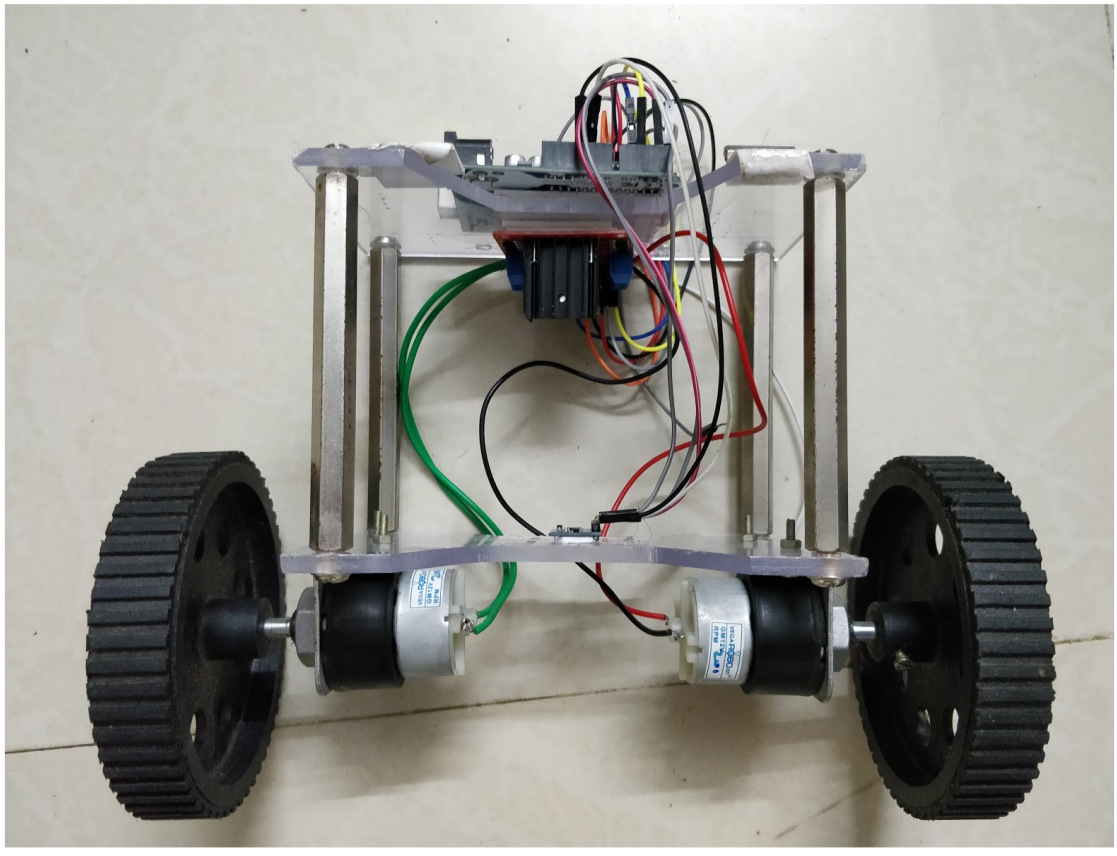
1. Acrylic Sheet (7mm)
2. 12 V DC Motors x 2
3. L-Clamps x 2
4. Nuts, screws and bolts
5. Double-sided adhesives
6. Arduino UNO Development Board
7. L298N Motor Controller Board- Datasheet L298N
8. MPU 6050 IMU Sensor- IMU Datasheet
9. 12V SMPS Wall Power Adapter
10. Connecting wires.

CONSTRUCTION:

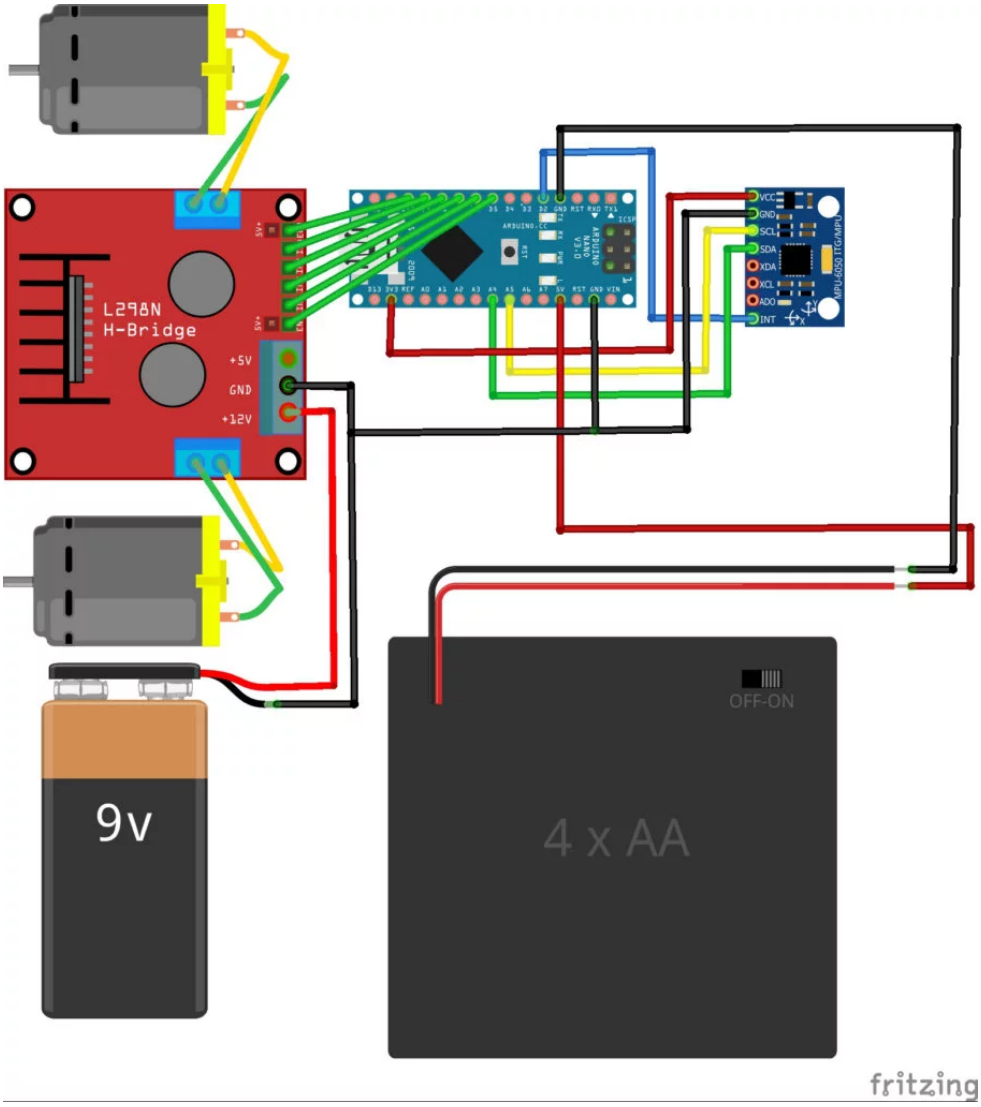








CIRCUIT DIAGRAM:

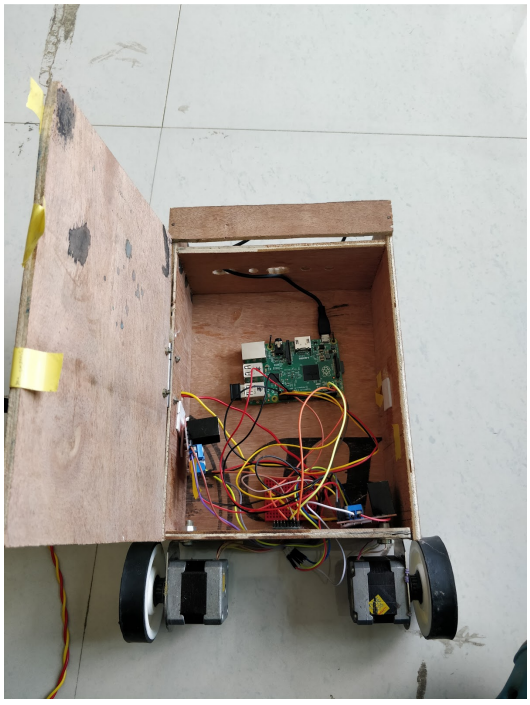

L298N
H-Bridge
+5V
GND
+12V
9v
4 x AA
OFF-ON
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT
fritzing

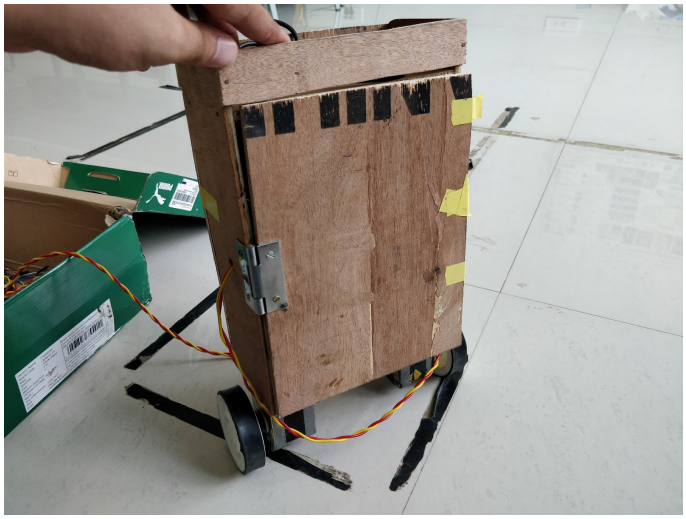
VERSION 2.0-

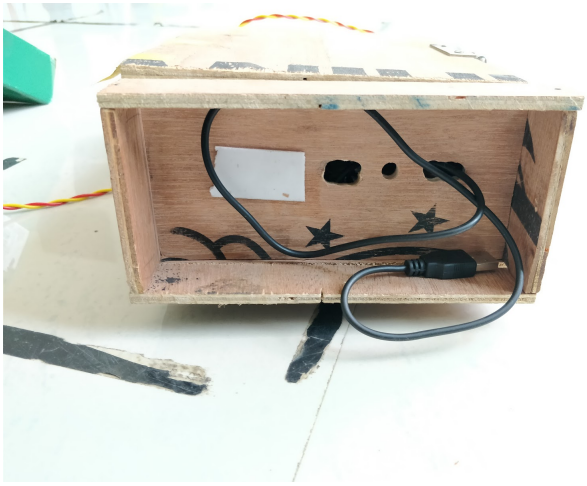
MATERIALS USED :

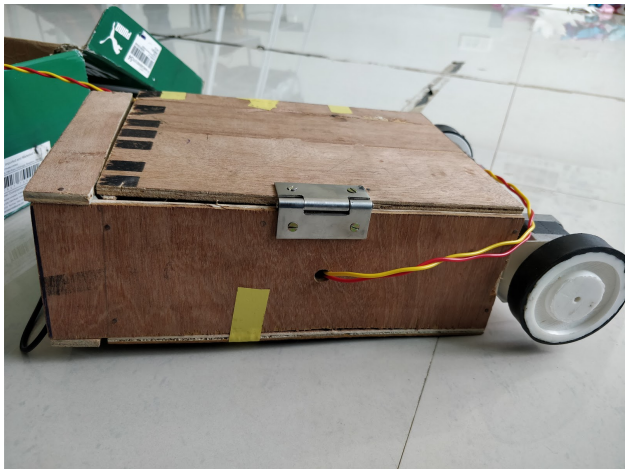
1. Plywood (Thickness = 6mm)
2. Nema-14 stepper motors - 2 - Nema 14 Datasheet
3. L-Clamps - 2
4. Nuts, screws and bolts
5. Double-sided adhesives
6. Raspberry Pi V3 B+ - Datasheet Pi
7. L298N Motor Controller Board- Datasheet L298N
8. MPU 6050 IMU Sensor- IMU Datasheet
9. 12V SMPS Wall Power Adapter
10. Connecting wires.

CONSTRUCTION:

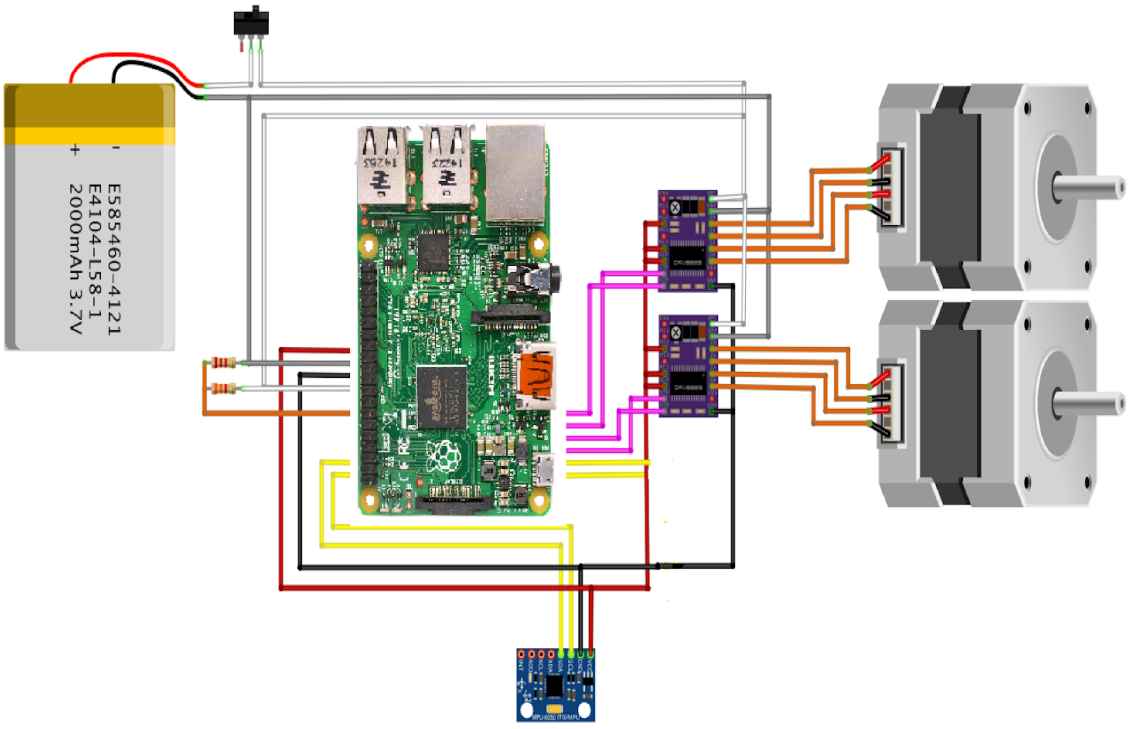




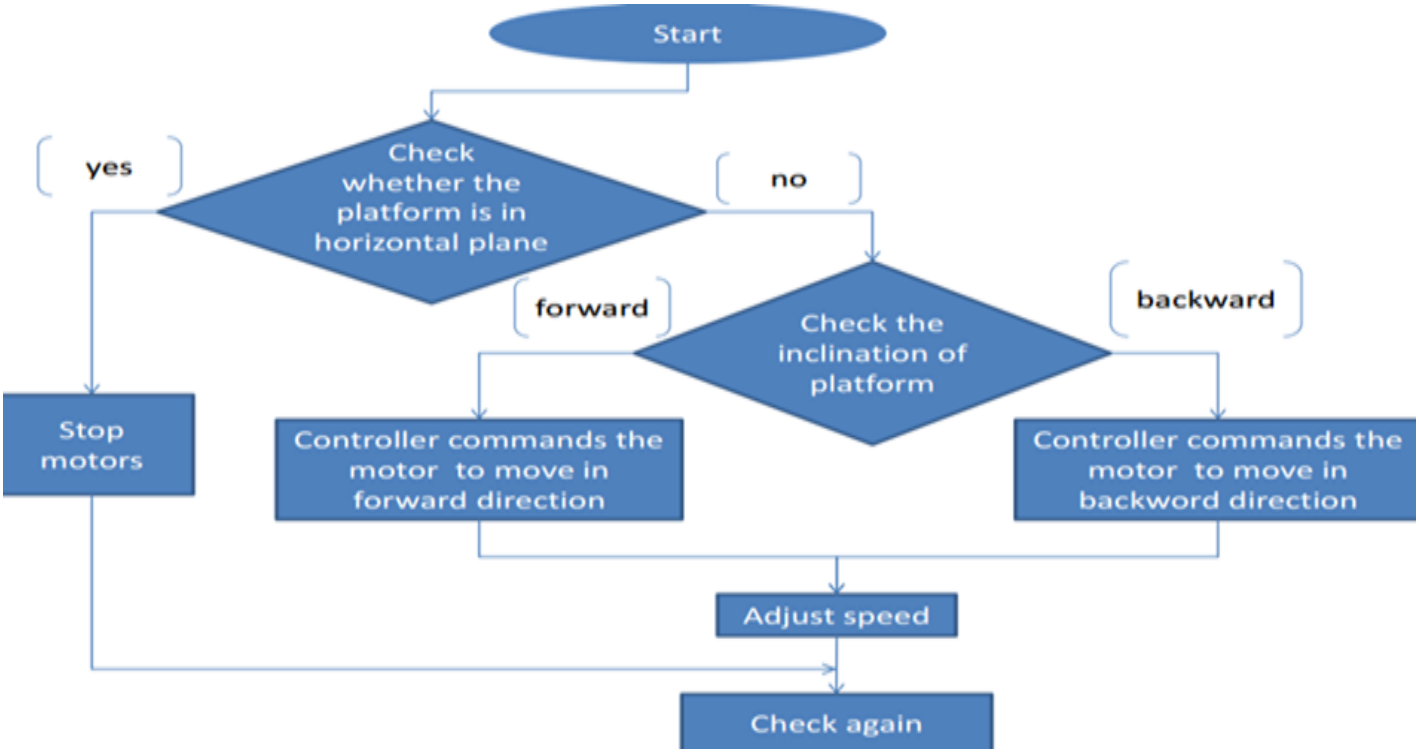




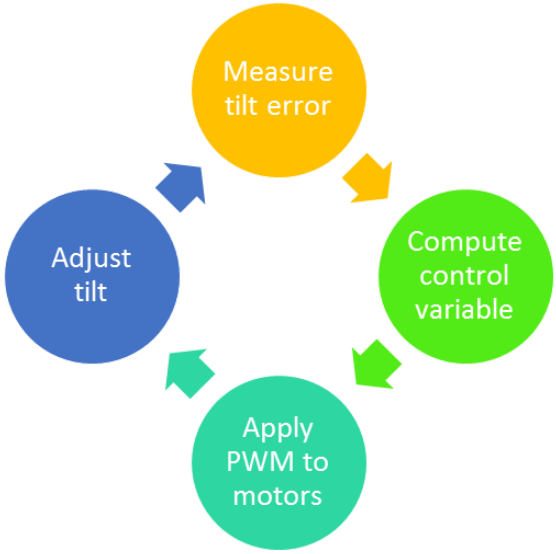
CIRCUIT DIAGRAM:



WORKING OF PLATFORM:



FLOW DIAGRAM FOR THE BALANCING ACTION OF THE PLATFORM



CALIBRATING YOUR MPU6050 MODULE

Before mounting the sensor we need to find the offset values of the raw data of accelerometer and gyroscope keeping the sensor in a perfect horizontal condition as

these values are needed while finding the robot's orientation angle. CODE LINK

FINDING THE ORIENTATION ANGLE FOR MPU'S RAW READINGS

The raw values from the sensor are of no use and hence need to be processed to get some useful data ie, orientation angle.Our bot and the PID algorithm only uses roll angle for stabilising.The algorithm for finding angle varies from using filters (kalman and complementary) to using quaternions. In our case we directly used the arduino code from the arduino website which uses quaternions. Code LINK.

MAIN CONTROLLING ALGORITHM ---THE PID CONTROLLER

The PID controller uses a feedback loop, which controls a parameter of the system (here angle) called the process variable (PV i.e., the roll angle in our case), which is supposed to match the desired output, the setpoint (SP i.e. vertical position = zero angle in our case.The PID controller uses an error(SP-PV), in each loop to control the system. It is more like a machine that takes angle deviation in each loop and gives the necessary signals to the motor. Each term of a PID algorithm has its own characteristics and roles, which are described in detail as followed:

P-Term

The P-Term makes an effort in proportion to how far the PV is from the SP at the present time. However, approaching closer to the SP, the error becomes so small that the controller cannot trigger the PV enough to catch the SP, => that there always exists a steady state error (SSE), which appears as an offset from the setpoint in the system.

However, a larger value of the proportional term can trigger the PV to the setpoint, but it makes the system unstable with oscillations and overshoots. Thus, the controller's response in such case behaves more like the response coming from an on/off controller. Thus, the P-controller alone is not sufficient for the most control designs.

$$\textbf{P-Term} = Kp.\ \textbf{e(t)}$$

where Kp = Proportional gain and e (t) = error at the present time "t".

I-Term

It takes into account all the errors present in the system from the starting point to that particular point of time in the process. It looks at the history of the error until that specific point.

The contribution from the I-Term tries to balance the difference in the time spent by the PV on the both sides of the setpoint, *i.e.* below and above the setpoint. For example, if PV spends 10 seconds running at 98% of the setpoint value, the I-Term will try to push the PV over to 102% for the other 10 seconds to compensate. Hence,

there is an overshoot. For any sensitive system, the overshoot can seriously damage the whole structure of the system.

$$\textbf{I-Term} = \int Ki.\ e(t)dt$$

where Ki = integral gain and τ = total time of operation of the controller.

D-Term

This takes the derivative of a PV of the system at every point. Thus, it predicts the future of the operation of the PID loop. The purpose of this term is to check how the process variable moves without overshooting the setpoint. The D-Term acts on the PI terms by counteracting them. As the PV approaches the setpoint, the PV settles with the set point with a small or no overshoot.

$$\textbf{D-Term} = Kd \cdot \textbf{de(t)/dt}$$

where, Kd represents the differential gain.

$$\text{The PID control signal} = Kp.\ \textbf{e(t)} + \int Ki.\ e(t)dt + K\textbf{d} \cdot \textbf{de(t)/dt}$$

PD controller code implementation

In our case, we implemented a pseudo pid code which is independent of the time domain and is a discrete type. The p term is simply multiplied with the error, I term is simply added to the error in each loop entry and d term is simply k_d multiplied with the previous and current error. So, each time we enter the loop the pid code gets automatically called off.

We have limited the PID values from -255 to +255 (it may vary in your case) and necessary offsets have been added so that the bot reaches the extreme values as soon as it suffers a small deviation and hence gives a quick response to any push. But in the bot it's quick responses are limited due to the use of low torque motors.

Corresponding to the PID values a PWM value is fed to the enable pins of the motors.

USING L298N MOTOR DRIVER

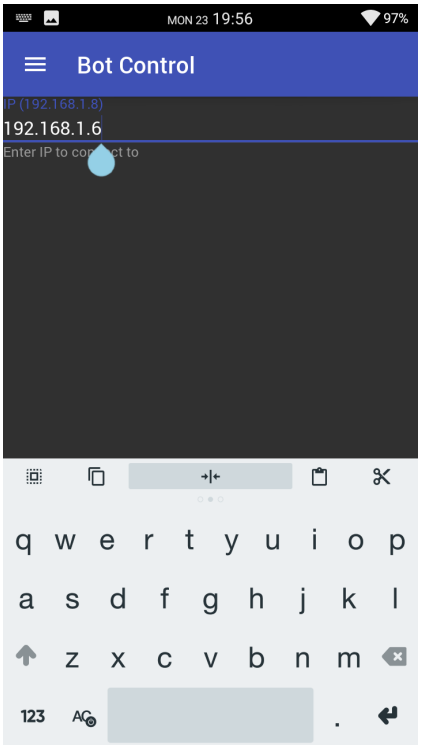
After interfacing the motors with the motor driver the jumper pins need to be removed from the enable pin only. They need to be connected to analog pins of arduino so as to give PWM signal (analogwrite) to the motors. **Always connect the ground of the power supply to the arduino ground for the motors to work.**

USING SMPS ADAPTER(12V , 2A)

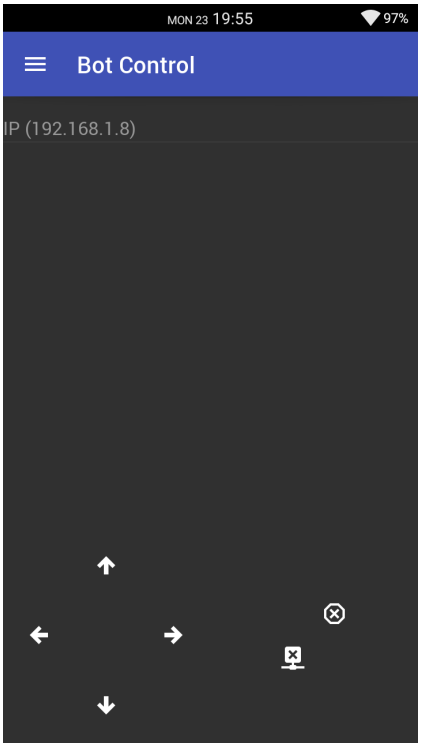
A SMPS adapter has been used so as to ensure a constant power supply unlike a DC

12V which is quite inconvenient to use.

APP CONTROL:



An Android app has been created that can send commands to the bot, however the integration of this with its PID control remains problematic and seems to be a thorny issue.Here are some screenshots:



ISSUES FACED AND IMPROVEMENTS TO BE DONE:

1. Lower torque performance of DC Geared motors and Steppers:

In order to increase the response to a harder push we can use **high torque motors** and for accurate control we can use **encoder based motors.`**

2. Sensor positioning:

We actually want to avoid as much of this translational movement as possible, as we are only interested in the rotation of the robot. Therefore **the sensor should be placed exactly on the axis of rotation**, between both wheels. Placing the sensor further up on the frame introduces noise and jitter into the readings

3. PID parameter tuning:

Identify the gains with lots of trials and identify the right constants. Varies from case

to case.

CONCLUSION:

We have successfully implemented and integrated - Sensor Data, Hardware, and Software to produce a Self Balancing Robotic Platform using the resources available to us.
We plan to improve upon this platform in the coming future to make it suitable for practical use for the application of **Hospital Supply Automation, warehouse automation and smart transportation applications.**

Links for Videos of it functioning are- Videos here

REFERENCES:

1. http://www.brokking.net/yabr_main.html - **Arduino based Self Balancing**

Robot

2. https://www.raspberrypi.org/blog/tag/robots/ - Great source
3. https://github.com/jarzebski/Arduino-MPU6050 - Arduino Library for interfacing with the IMU. Code was adapted to Pi from this
4. https://www.youtube.com/watch?v=-bQdrvSLqpg - A comprehensive YouTube video that goes through the intricacies of PID Tuning
5. http://abyz.me.uk/rpi/pigpio/ - A Python/C/C++ Library for the Raspberry Pi