Balance On Wheels

Project Report

Eklavya Mentorship Programme

SOCIETY OF ROBOTICS AND AUTOMATION,

VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE, MUMBAI

OCTOBER 2022

Acknowledgement

We had a great time learning under our seniors. They guided us well and answered every question of ours along the way. We are grateful to every effort of theirs to make us familiar with the topic and provide additional resources along the way. They not only taught us concepts related to the project but also skills which would help us a long way in our engineering career. We would like to thank **SRA VJTI** for giving us this golden opportunity and our seniors who took the time and efforts to guide us.

Special thanks to our mentors **Marck Koothoor**, **Ayush Kaura** and **Priyal Awankar** for guiding us through the entire duration of the project and helping us achieve our goal.

Arsh Khan
khanarsh0124@gmail.com
+91 9987408678

Shreyas Patil
patilshreyasvjti@gmail.com
+91 9769983689

TABLE OF CONTENTS:

1 . PROJECT OVERVIEW

1.1 PROJECT IDEA:

Modelling and Control of Two-Legged Wheeled Robot: In this project, we will be building a two wheeled robot (with a hip and knee joint) which can balance at variable heights including a provision for jumping.

1.2 TECHNOLOGIES AND TOOLS USED:

1. SolidWorks:
SolidWorks is a solid modelling computer-aided design (CAD) and computer-aided engineering (CAE) computer program. Here it’s used to design the Robot.

2. Autocad:
AutoCAD is a commercial computer-aided design (CAD) and drafting software application. Here it’s used to edit the dwg file for Laser-Cutting.

3. C
C is the programming language in which we have implemented all the algorithms of Self balancing, Servo Locking and Jumping.

4. Python and Matplotlib
Python programming scripts were used to calculate the parameters of our Bot. Matplotlib was used to plot the graphs to find the extension of the leg on changing the parameters of the bot.

5. Creality
Creality software was used for 3D Printing some parts of the bot.

6. ESP-IDF Framework (v4.4)
ESP-IDF(Espressif IoT Development Framework) is the official software development environment for the hardware based on the ESP32 chip by Espressif. We use ESP-IDF for menu configuration, then for building and flashing firmware onto an ESP32 board.

2. INTRODUCTION

2.1 BASIC PROJECT DOMAINS:

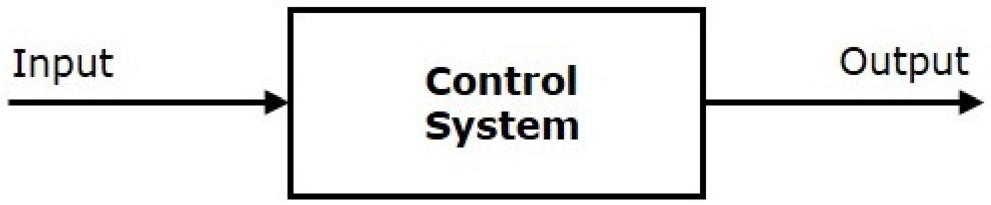
The main domains of the project are CAD Design and Modelling, Embedded and Control Systems.

2.2 THEORY

The designing of the robot requires a good amount of knowledge of SolidWorks. The dimensions of the bot have to be precise for it to work properly. A thorough understanding of the PID control system is very important to implement the Self Balancing algorithm at variable heights. Also, one must know C as the algorithms need to be written in the language.

2.3 Basics Of Control System

A control system is a system, which provides the desired response by controlling the output. The following figure shows the simple block diagram of a control system.



The main feature of a control system is that there should be a clear mathematical relationship between the input and output of the system.
There are two main types of control systems. They are as follow-

1. Open-loop control systems
2. Closed-loop control systems

Open Loop Control System

A control system in which the control action is totally independent of the output of the system, then it is called an open-loop control system.

Closed Loop Control System

Control systems in which the output has an effect on the input quantity in such a manner that the input quantity will adjust itself based on the output generated is called a closed-loop control system.

2.4 Why Use PID

PID-control is most commonly used because it combines the advantages of each type of control. This includes a quicker response time because of the P-only control, along with the decreased/zero offset from the combined derivative and integral controllers.

It is much more practical than the typical on/off controller, PID controllers allow for much better adjustments to be made in the system. While this is true, there are some advantages to using an on/off controller including that they are (1) relatively simple to design and execute and (2) binary sensors and actuators (such as an on/off controller) are generally more reliable and less expensive.

3. STAGES OF PROGRESS

3.1 <u>Solidworks Basics</u>

Parts are the basic building blocks in the SOLIDWORKS software. Assemblies contain parts or other assemblies, called subassemblies. A SOLIDWORKS model consists of 3D geometry that defines its edges, faces, and surfaces.
The SOLIDWORKS software lets you design models quickly and precisely. SOLIDWORKS models are:

- Defined by 3D Design
- Based on components

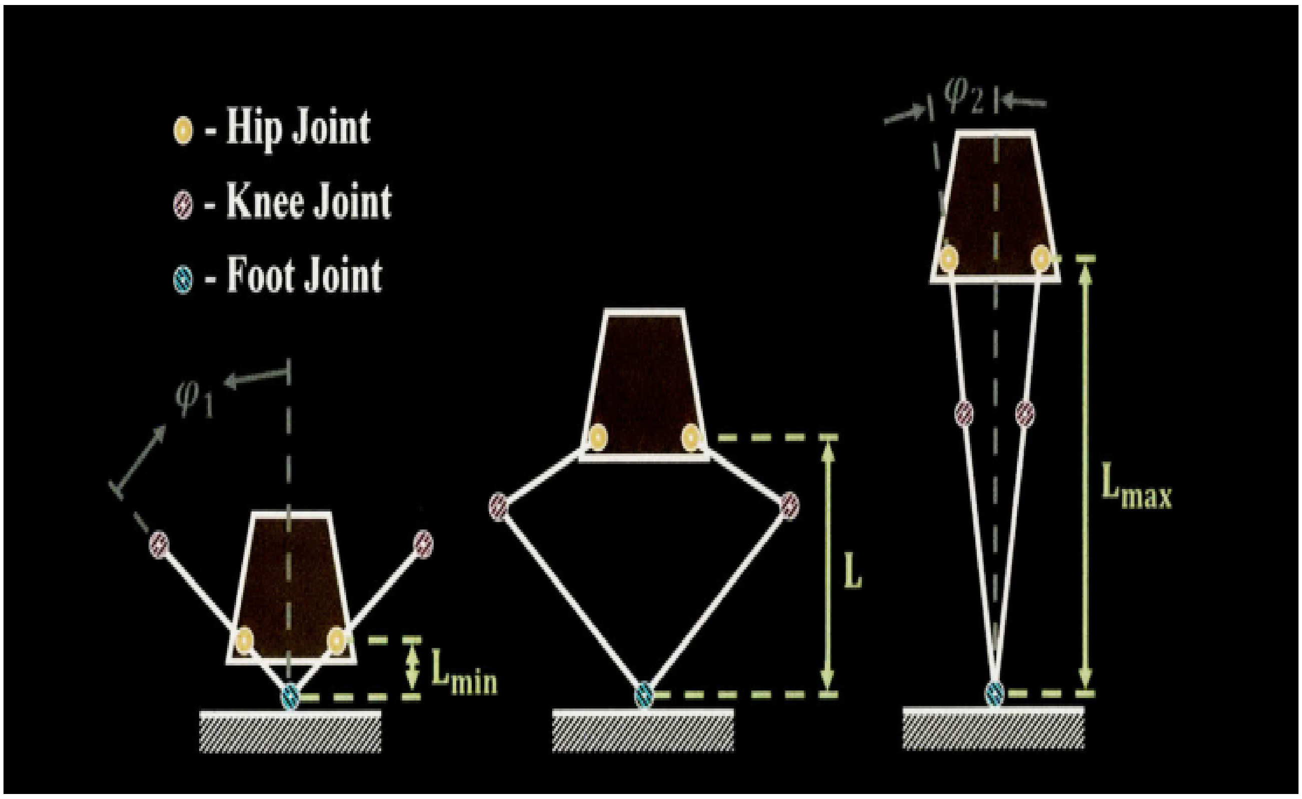
SOLIDWORKS uses a 3D design approach. As you design a part, from the initial sketch to the final result, you create a 3D model. From this model, you can create 2D drawings or mate components consisting of parts or subassemblies to create 3D assemblies. You can also create 2D drawings of 3D assemblies.
When designing a model using SOLIDWORKS, you can visualise it in three dimensions,the way the model exists once it is manufactured.

3.2 <u>Basics of Design of Bot</u>

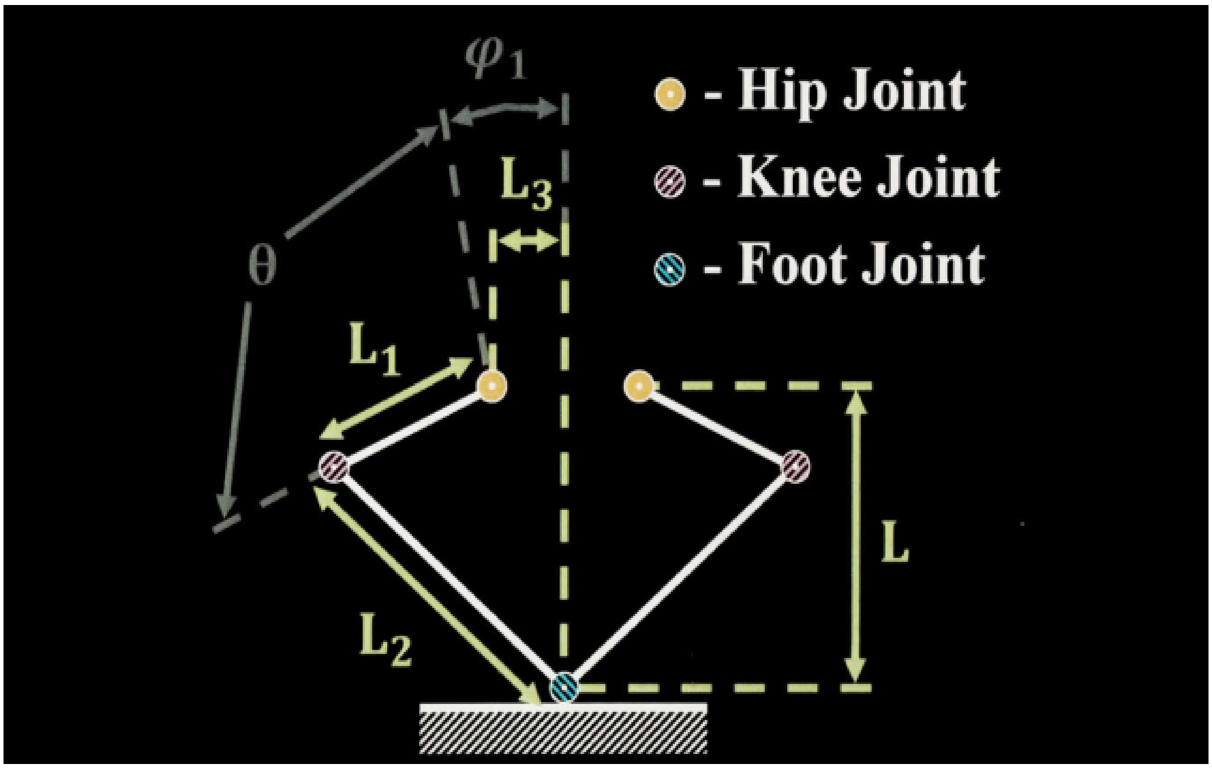
We designed a Bi-Pedal BOT on the Pantograph Mechanism, which is a mechanical linkage connected in a manner based on parallelograms.

The Kinematics of the Bot were derived for Pantograph design from which the key parameters such as the length of the linkages were able to be modified and optimised during the design process.

The Hip Joint is defined to rotate in a 180 degree arc which results in the legs moving between Parallel and Antiparallel configurations. The angle between the vertical and the legs is shown for both the parallel and anti-parallel configurations and defined as the angles **Φ1** and **Φ2** as shown in Figure. The trapezoid which represents the body of the robot.

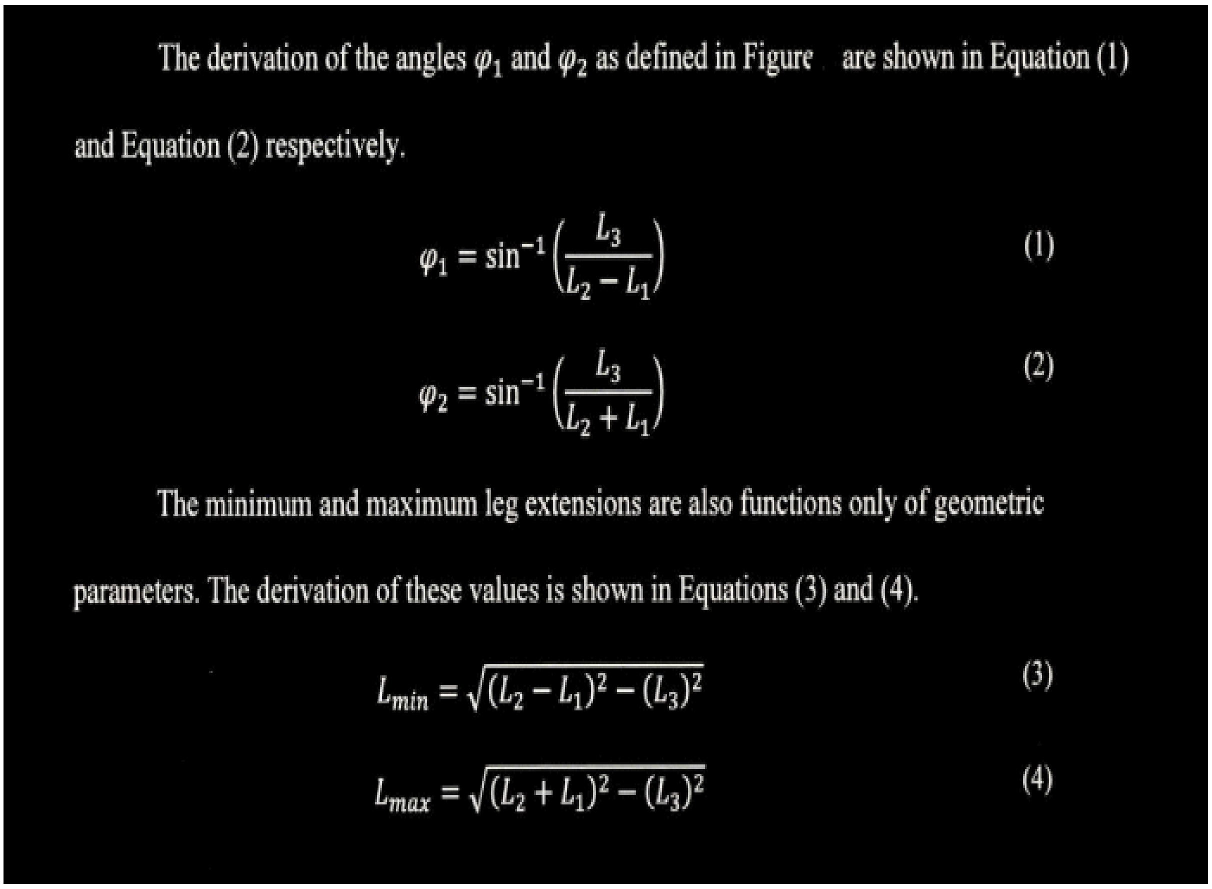


In the above Figure Leg extension is defined as the variable L is the vertical distance between the axis of the hip joints and the foot joint. The leg is shown at minimum extension Lmin in the antiparallel configuration and at maximal extension Lmax in the parallel configuration.



The above figure shows when the bot is at an arbitrary position. Here,
L1 → Length of Upper Leg Segment (From Hip Joint to Knee Joint)
L2 → Length of Lower Leg Segment (From Knee Joint to Foot Joint)
L3 → Length from the Hip joint to the Mechanisms axis of Symmetry

Equations Used To Derive the System Design Variables



The derivation of the angles φ_1 and φ_2 as defined in Figure are shown in Equation (1) and Equation (2) respectively.

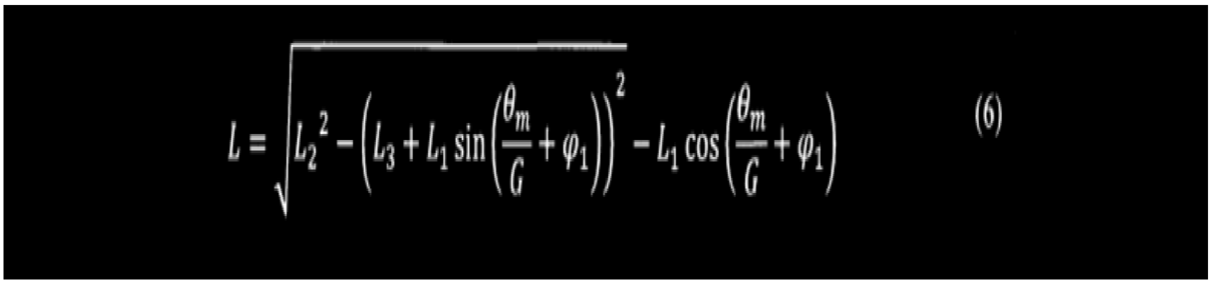
$$\varphi_1 = \sin^{-1}\left(\frac{L_3}{L_2 - L_1}\right) \qquad (1)$$

$$\varphi_2 = \sin^{-1}\left(\frac{L_3}{L_2 + L_1}\right) \qquad (2)$$

The minimum and maximum leg extensions are also functions only of geometric parameters. The derivation of these values is shown in Equations (3) and (4).

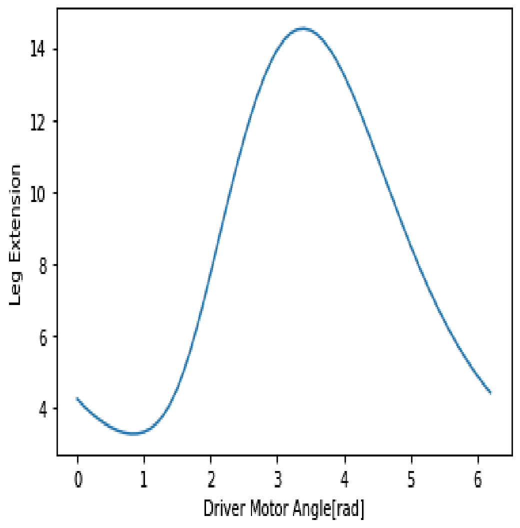
$$L_{min} = \sqrt{(L_2 - L_1)^2 - (L_3)^2} \qquad (3)$$

$$L_{max} = \sqrt{(L_2 + L_1)^2 - (L_3)^2} \qquad (4)$$



$$L = \sqrt{L_2^{\,2} - \left(L_3 + L_1 \sin\left(\frac{\theta_m}{G} + \varphi_1\right)\right)^2} - L_1 \cos\left(\frac{\theta_m}{G} + \varphi_1\right) \qquad (6)$$

In Above Image we have equations to calculate the **Φ1**,**Φ2**, L1, L2 and L3 . Also In above Image at end , it is the General Equation of Length (vertical distance between the axis of the hip joints and the foot joint) which was used to plot the graph between the leg extension vs change in driver motor angle which is given below. Here, G (Gear Ratio) which is taken to be 1 for our Bot.

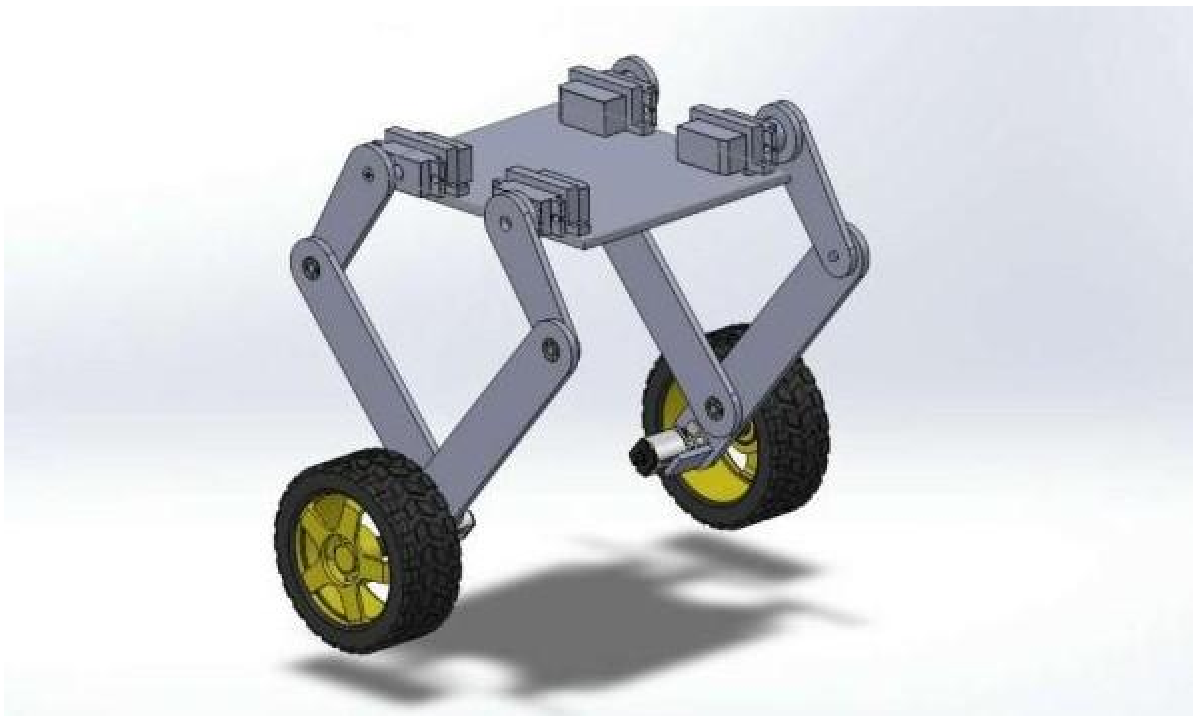


The above Graph gives is about change of Leg Extension with respect to change in Driver Motor Angle. At the start or at the end the slope is nearly equal to zero. In the middle, the slope is large. This means that small changes in motor angle cause a large amount of leg extension.

3.3 <u>Design of Bot in Solidworks</u>

Solidworks design consists of 3D modelling of Bot design. We needed to check if various other aspects of systems and mechanical design satisfy our requirements. So CAD design forms a very important part of our project.

3.4 CAD Model of Bot



3.5 Hardware

Microcontroller

In this project we are using ESP32 microcontroller which is a series of low-cost, low-power system on a chip microcontrollers with integrated Wi-Fi and dual-mode Bluetooth. ESP-IDF is the official framework and compiling and flashing code in the microcontroller

Motor Drivers

In this project we are using TB6612FNG Motor Drivers. They are simply current amplifiers. The provide the motors with a voltage of 12V.

Motion Processing Unit (MPU)

In this project we are using Mpu6050 sensor which has a 6-axis motion tracking
This sensor comprises both 3 axis accelerometer and 3 axis gyroscope. These sensors together give the roll, pitch, yaw values of the bot.It is an integral part of Control Systems.

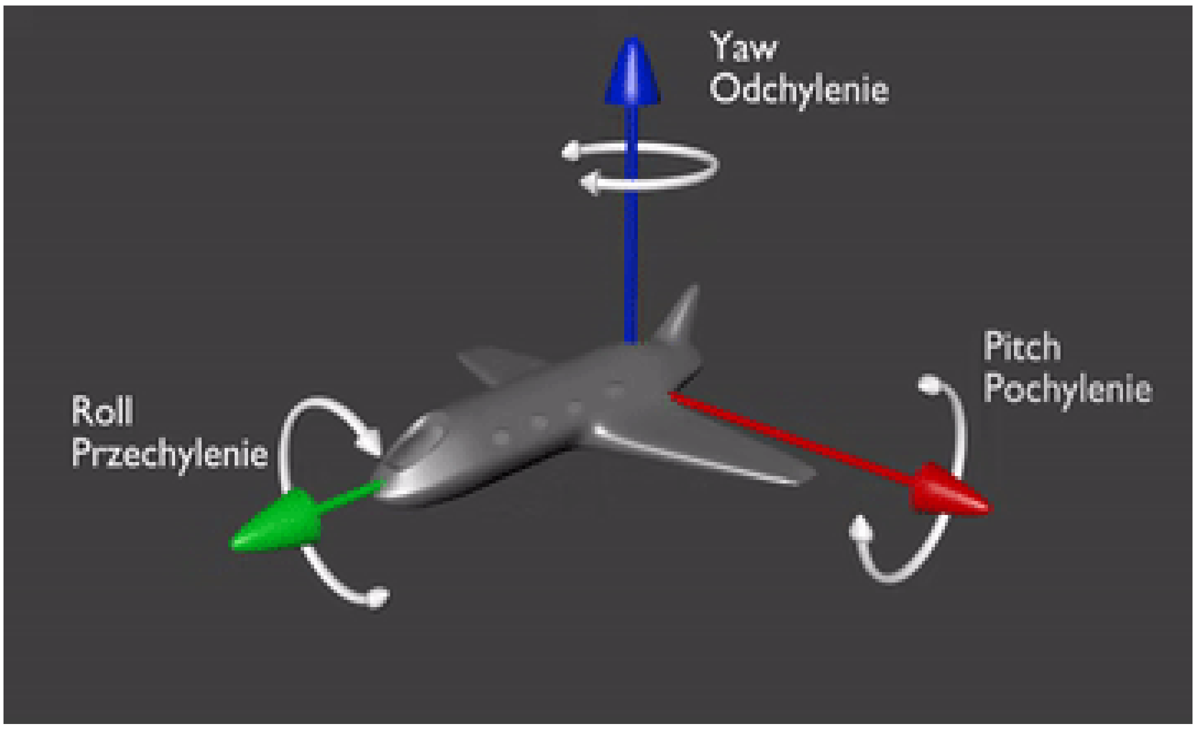
N20 Motors

In this project we are currently using 6V 600 rpm N20 motors

Servo Motors

In this project we are using metallic MG90 Servos.

3.6 Roll, Pitch, Yaw



Roll

Roll is the angle made by the bot with X-axis.

Pitch

Pitch is the angle made by the bot with Y-axis.

Yaw

Yaw is the angle made by the bot with Z-axis.

3.7 Understanding Complementary Filter

The Mpu6050 for roll,pitch,yaw values.Euler angles are complementary roll,pitch,yaw values.

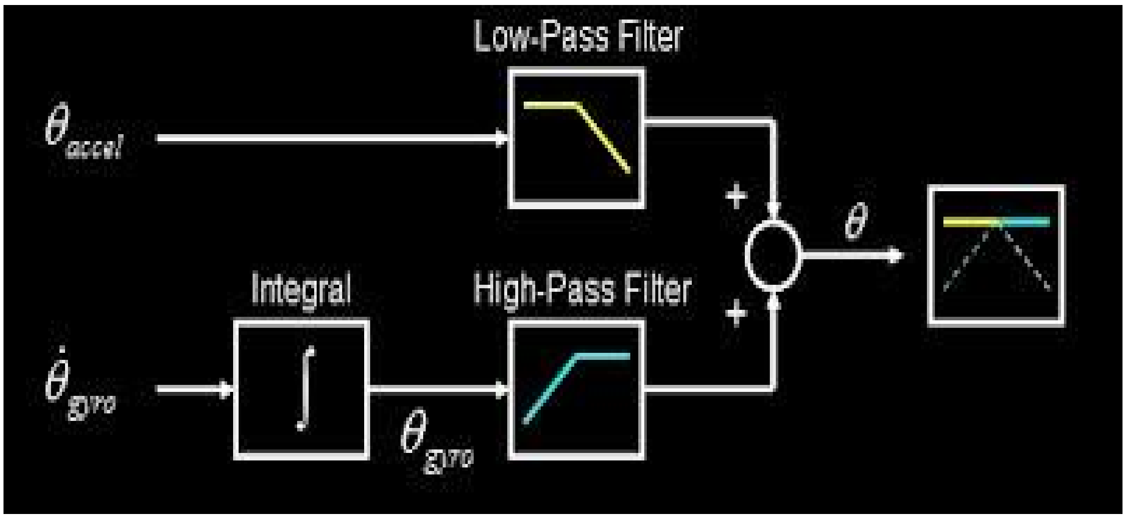
We get the readings of angle from both Gyroscope and Accelerometer.Both sensors can give readings directly.

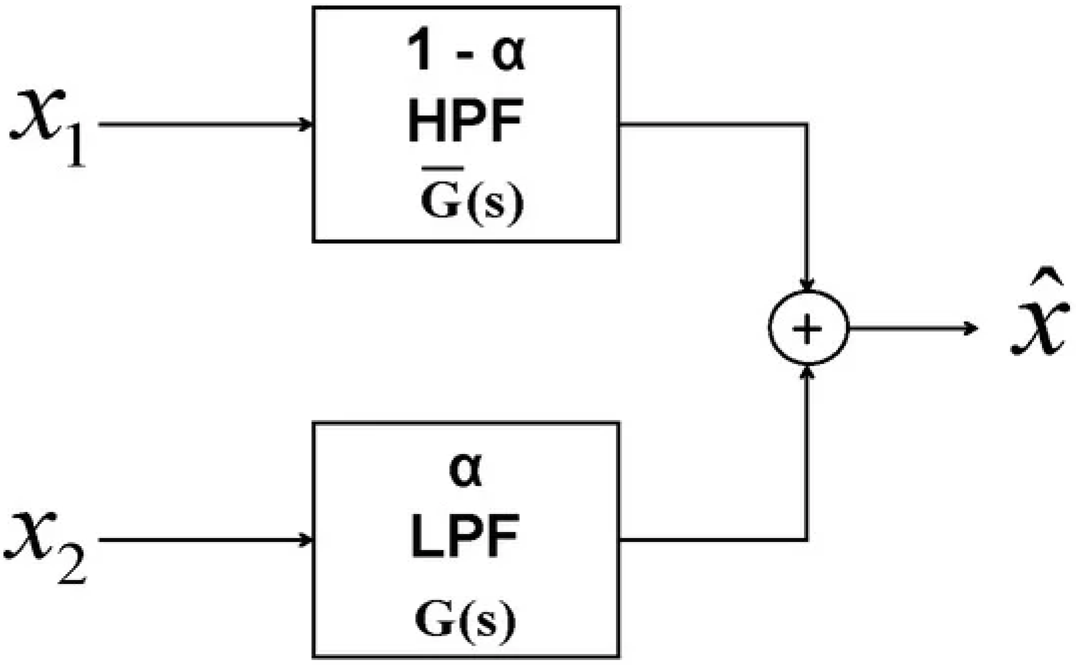
Gyroscope has a problem of Gyroscopic Drift which accumulates over time, so it gives reliable readings for a short period of time.

Whereas Accelerometer is jittery and sensitive, but the readings are based on g-vector which has an absolute value, the readings can be relied on for long term.

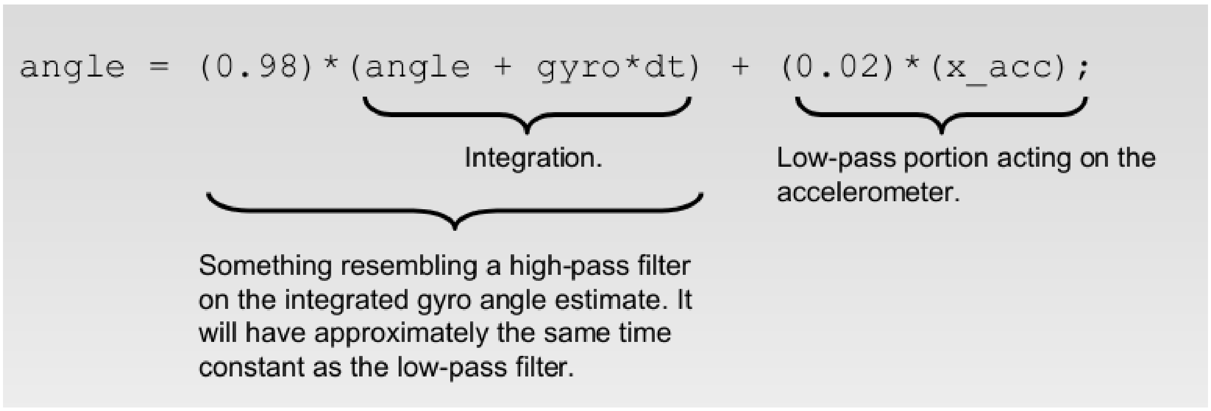
To use the benefits of each Sensors we do Sensor fusion.

Idea behind the Complementary filter is to take slow moving signals from gyroscope and fast moving signals from an accelerometer and combine them. Accelerometer data is reliable for long term so we apply a "Low pass" filter to it whereas Gyroscope data is reliable in short term so we use "High pass" filter.





Complementary Filter Code



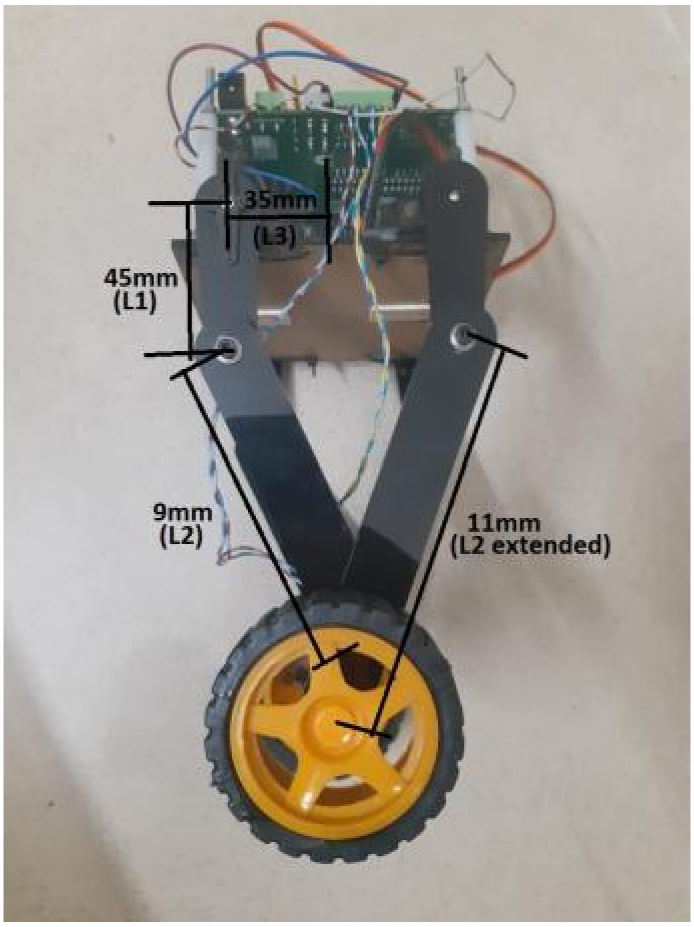
```
angle = (0.98)*(angle + gyro*dt) + (0.02)*(x_acc);
```

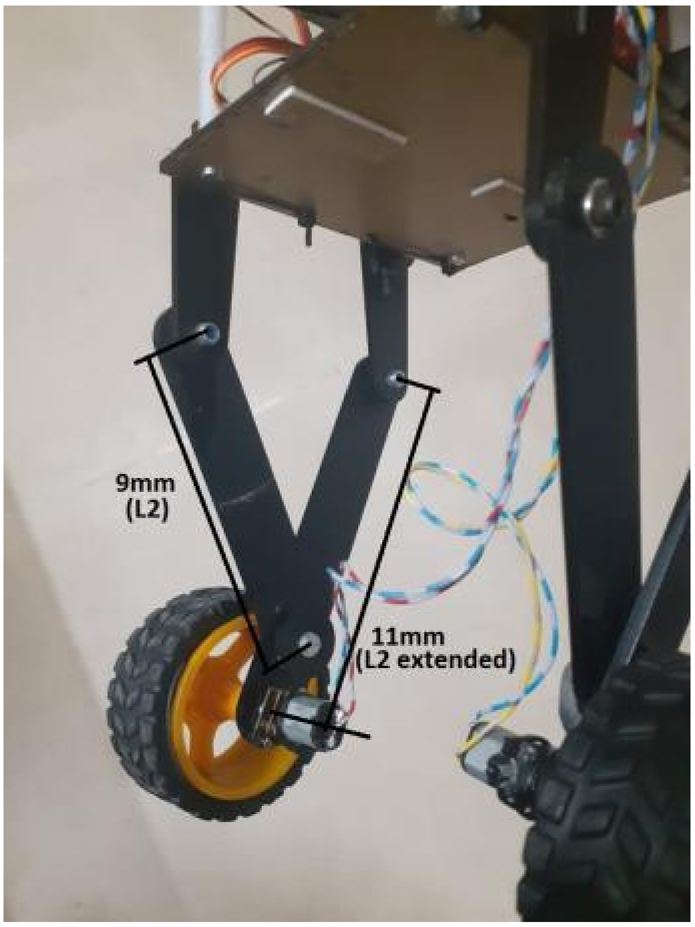
Integration.

Low-pass portion acting on the accelerometer.

Something resembling a high-pass filter on the integrated gyro angle estimate. It will have approximately the same time constant as the low-pass filter.

3.8 Hardware Bot

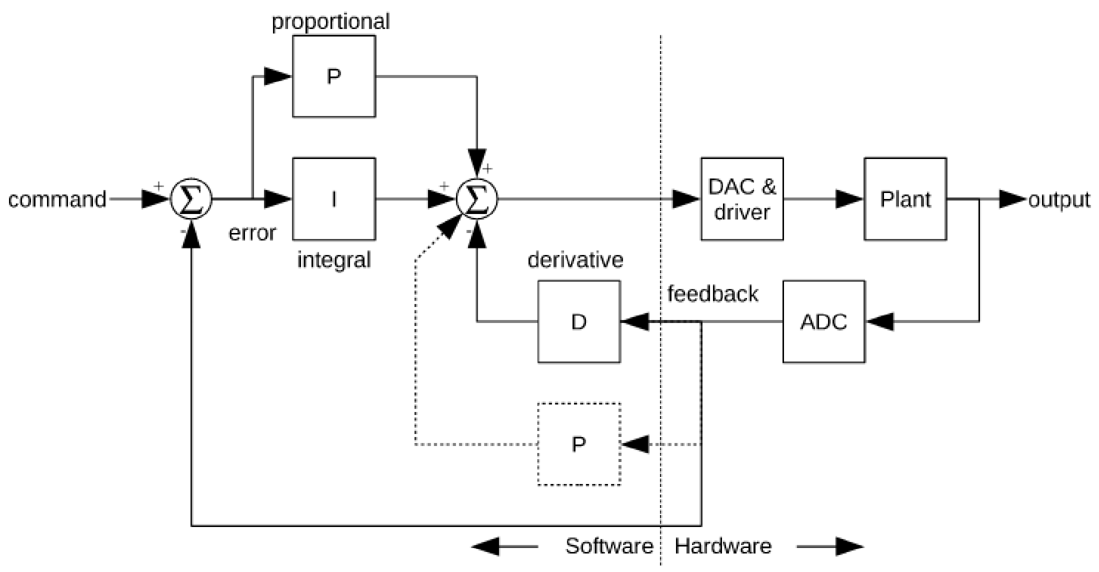




4. IMPLEMENTING THE ALGORITHMS

4.1 Understanding PID

The "PID" in "PID Control" stands for **"Proportional, Integral, Derivative"**. These three terms describe the basic elements of a PID controller. Each of these elements performs a different task and has a different effect on the functioning of a system and it is one kind of device used to control different process variables like pressure, flow, temperature, and speed in industrial applications.



<u>P Term (Proportional Term)</u>

The PROPORTIONAL value is directly proportional to the position of the object with respect to the reference.

U α e

∴ U = K .e

Where, K = Proportional gain e = error

U = Correction Term

In our Bot the Error is the Pitch Error, i.e, difference between current pitch angle to the initial angle of reference (Set point).

Larger values of Kp typically mean faster response since the larger the error, the larger the Proportional term correction signal. However, an excessively large proportional gain will lead to instability and oscillation.

<u>D Term (Derivate Term or Damping term)</u>

The DERIVATIVE is the rate of change of the proportional value.

$U_2 \propto$ rate of change of error

∴ U_2 = Kd.(de/dt)

Where, Kd = Derivative gain

de/dt = Change in Error = (Current error - Previous error)/Time

A derivative term does not consider the magnitude of the error. A pure 'D' controller takes the rate of change of error, and tries to bring this rate to zero. It aims at flattening the Error Trajectory into a horizontal line, Damping the Force applied , and so reduces Overshoot (Reduces Oscillations occurred by P Term).

<u>I Term (Integral Term or Steady State Error)</u>
The INTEGRAL value records the history of your Bot's motion. It is a sum of all the values of the Proportional term that were recorded.

U3 α Cumulative Error
∴ U3 = Ki * Cumulative Error
∴ U3 = Ki * ∫e(t)dt

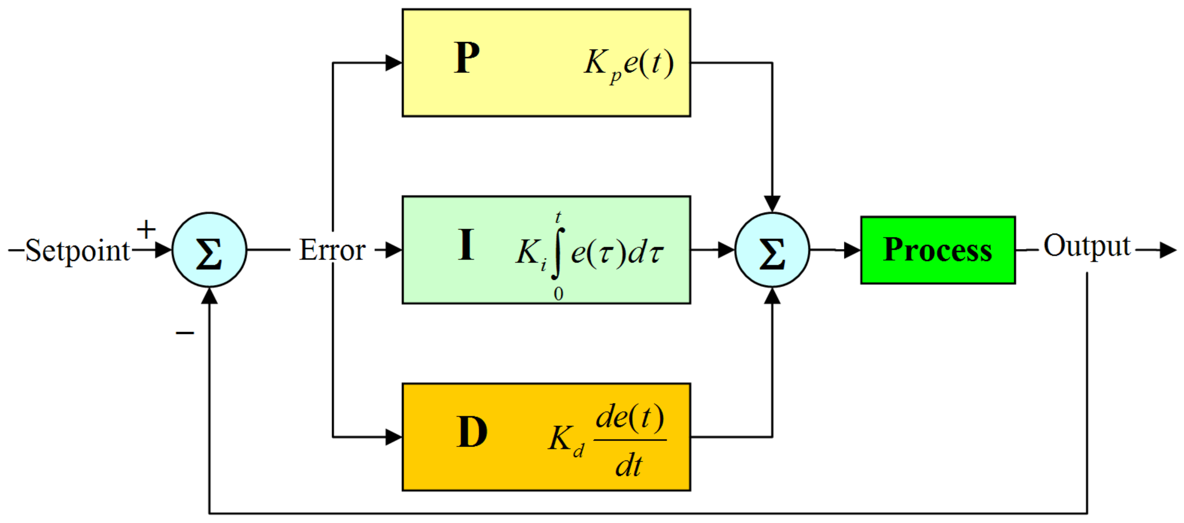
Where, Ki = integral gain
∫e(t)dt = Cumulative error (in time interval 0 to t)

Integral term increases action in relation not only to the Error but also the time for which it has persisted.

So, if the Applied Force is not enough to bring the error to zero, this force will be increased as time passes.
Applying too much integral when the error is small and decreasing will lead to overshoot.

Therefore,Total Error is the summation Error of P,I,D term :

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt}$$



4.2 <u>Self-Balancing Algorithm</u>

For self-balancing the bot, we need to keep the bot a bit above our desired angle (Setpoint), since the bot is very fast and will fall down once it's past the desired angle. The pitch angle is required for Self-Balancing which is obtained from the Inertial Measurement Unit(IMU sensor) plug-in.

Firstly we initialise Euler_angle and mpu_offset to zero. Euler angles are complementary roll,pitch,yaw values.We get the readings of angle from both Gyroscope and Accelerometer.Both sensors can give readings directly. We do Sensor fusion of both the sensors using complementary filter, which gives accurate readings.

Next we read the Pitch value of angle from MPU 6050.

```
pitch_angle = euler_angle[1]
```

We also read Setpoint from Https server.

```
pitch_cmd = read_pid_const().setpoint
```

Then we calculate the Pitch Error by subtracting pitch angle from setpoint of bot.

```
pitch_error = pitch_cmd - pitch_angle
```

Then we pass the current generated error to `error_calculation function(pitch_error)`which returns absolute pitch correction term for motor command

Calculation of Error :

Firstly in `error_calculation()` function we find Difference in Pitch Error which is the difference of current pitch error and previous pitch error.

```
pitch_difference_error = current_pitch_error -
prev_pitch_error
```

Then we calculate total pitch error which is useful for Steady state error(Integral Term)

```
total_pitch_error =total_pitch_error +
current_pitch_error
```

Then we calculate the rate of change of error for Derivative 'D' term to reduce the oscillations caused by P term.Here dt =1 sec

```
pitch_rate = pitch_difference_error/dt
```

For Calculation of PID terms

For P term we read the value of kp from https server and multiply that value with current pitch error.

```
P_term = read_pid_const().kp * current_pitch_error
```

For D term we read the value of kd from https server and multiply it by bounded value of Pitch Rate between MIN and MAX Pitch Rate predefined.

```
D_term = read_pid_const().kd * bound(pitch_rate,-MAX_PITCH_RATE,MAX_PITCH_RATE)
```

For I term we again read the value ki from https server and multiply it by bounded value of total pitch error between MIN and MAX Pitch correction.

```
I_term = read_pid_const().ki *
bound(total_pitch_error,-MAX_PITCH_CORRECTION,MAX_PITCH_CORRECTION)
```

Now that we have calculated PID error terms we then calculate the pitch correction term which is sum of all the errors.

```
pitch_correction = P_term + I_term + D_term
```

Then we plot the graph of PID term, pitch correction and current pitch error terms onto the server.

```
plot_graph(P_term,D_term,I_term,pitch_correction,
current_pitch_error)
```

Then we store the current pitch error as previous pitch error for further calculations when this error calculation function runs again.

```
prev_pitch_error = current_pitch_error
```

Then we calculate the absolute pitch correction term and then returing this term back to main loop

```
abs_pitch_correction = fabs(pitch_correction)
```

The received absolute pitch error is passed onto the motor command function which will decide the motion of bot to counter the error.

```
motor_command(abs_pitch_correction,pitch_error,&motor_cmd,&motor_duty_cycle)
```

The function takes in 4 parameters. absolute pitch correction term received from error_calculation function, current pitch error, motor command(calculated theoretical calculated error values obtained by PID) and motor duty cycle (which gives proper actual velocity corrections to motors obtained from motor command).

In `motor_command()` function :

Firstly we calculate motor command which is the bounded value of absolute pitch correction between 0 and Max pitch correction.

```
*motor_cmd =
bound(abs_pitch_correction,0,MAX_PITCH_CORRECTION);
```

Then we calculate the motor duty cycle which is the bounded value of motor command between MIN and MAX PWM which are predefined in code.

```
*motor_duty_cycle =
bound(*motor_cmd,MIN_PWM,MAX_PWM)
```

The Main Part of Self Balancing Code :

Now If the pitch_error is -ve it implies that the front of the bot is below the the desired position (i.e. pitch_angle < pitch_cmd),in simple words Bot is tilting downwards so both the motors are given command to rotate in forward direction to balance itself.

```
if(pitch_error<-1)
{
set_motor_speed(MOTOR_A_0,MOTOR_FORWARD,*motor_dut
y_cycle);
set_motor_speed(MOTOR_A_1,MOTOR_FORWARD,*motor_dut
y_cycle);
}
```

If the pitch_error is +ve it implies that the front of the bot is above the the desired position (i.e. pitch_angle > pitch_cmd),in simple words Bot is tilting Upwards so both the motors are given command to rotate in backward direction to balance itself.

```
if(pitch_error>1)
{
set_motor_speed(MOTOR_A_0,MOTOR_BACKWARD,*motor_du
ty_cycle);
set_motor_speed(MOTOR_A_1,MOTOR_BACKWARD,*motor_du
ty_cycle);
}
```

If the pitch_error is equal to zero , it implies that the front of the bot is exactly in equilibrium position which is neither tilting upwards and downwards (i.e. pitch_angle = pitch_cmd),in simple words Bot is equilibrium it has achieved Self Balancing , now both the motors are given command to Stop by giving them Zero Duty Cycle.

```
if(pitch_error==0)
    {
        set_motor_speed(MOTOR_A_0,MOTOR_STOP,0);
        set_motor_speed(MOTOR_A_1,MOTOR_STOP,0);}
```

`set_motor_speed()` function takes in 3 Inputs.Firstly Motor Id,Secondly Motor Rotation Direction and Lastly Duty Cycle.

And all the above functions continue to execute again in a loop for achieving the desirable result,i.e, Self Balancing.

4.3 <u>**Self Balancing and Servo Lock Algorithm**</u>

This is the algorithm of integrating Self Balancing Code with servos locking up at minimum height for Self Balancing of Bot at Minimum Height. PID is used for balancing along with servo set zero which servos lock the bot in minimum height simultaneously so that the bot can remain stable at that position or otherwise the bot may collapse. Also before Balancing the bot we set servo zero first and then the self balancing code starts to function.

Servo Lock code is given priority 1 and Self-Balancing is given priority 2 . In Self-Balancing the bot achieves to remain in a balanced state and its algorithm is written above in 4.2.

Servo Lock code sets the motor at a particular angle, fixing the leg of the bot at that angle which possibly cannot be moved or changed.

In Servo Lock code firstly we do servo configurations of all the servos used . In configure servo_pin It contains the gpio pin number to which servo is connected. It has 7 parameters

`min_pulse_width`:It contains the minimum pulse width of servo motor

`max_pulse_width` : It contains the maximum pulse width of servo motor

`max_degree` : It contains the maximum degree servo motor can rotate `mcpwm_num` : It contains MCPWM unit to use

`timer_num` : It contains MCPWM timer to use

`gen` : It contains MCPWM operator to use

Example :

```
servo_config servo_a = {
    .servo_pin = SERVO_A,
    .min_pulse_width = 500,
    .max_pulse_width = 3000,
    .max_degree = 180,
    .mcpwm_num = MCPWM_UNIT_0,
    .timer_num = MCPWM_TIMER_0,
    .gen = MCPWM_OPR_A,
};
```

Then we start the servos by running

```
enable_servo()
```

which enables the Servo port on the sra board and sets up PWM for the three pins in servo port.

Then,to change the angles of servo we use the function

```
set_angle_servo(servo_config *config, unsigned int degree_of_rotation)
```

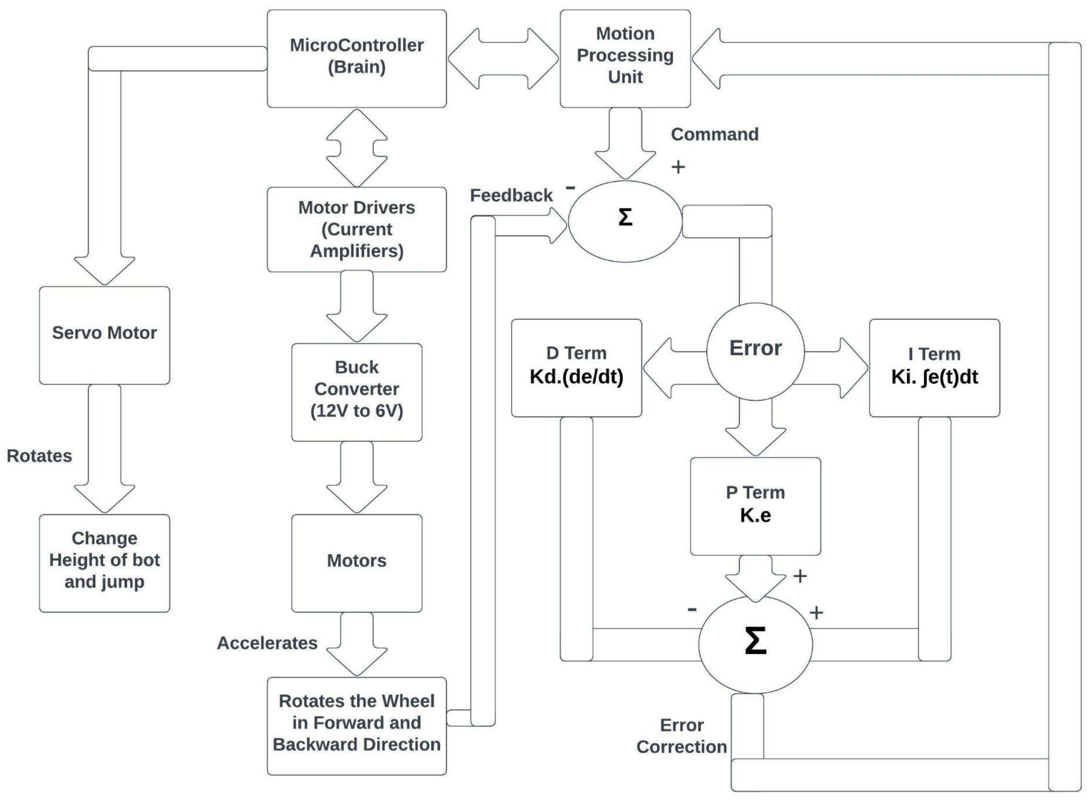
is used to set the angle of the servos attached to the servo port of SRA Board. It takes 2 parameters

`config` it is the pointer to the servo_config struct

`degree_of_rotation` angle to which the servo must be set, depends on value of MAX_DEGREE macro

Therefore both servo locking code and self balancing code runs together to balance the bot at any configuration.

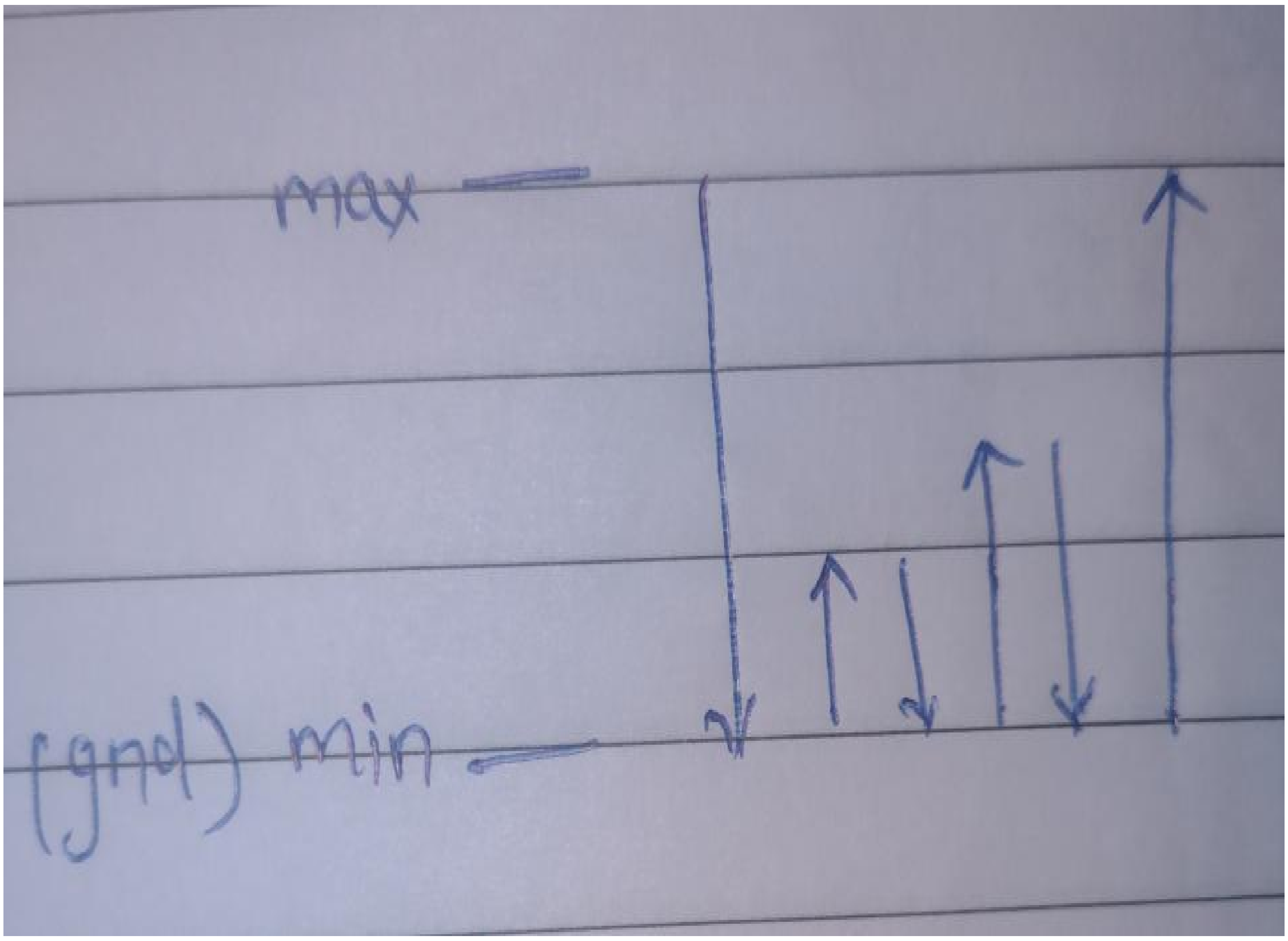
Flowchart of Algorithm



Note : To balance the bot at variable height, we need to tune the Kp,Ki,Kd values of the bot respectively for each height. The change in these values will be very small, max ± 10-20% from the values tuned at minimum and maximum height.

4.4 **Jumping Algorithm**

In Jumping Algorithm our main goal is that the bot is able to lift itself up from the ground by using the combinations of servo angles.



Here, basically we are increasing the height of the robot in iterations rather than in a single movement. This is because such movement with appropriate delays produces a thrust which pushes the bot upwards. Such thrust is not produced as a single down-up movement.

4.5 Self-Balancing, Servo Lock and Jumping

In this algorithm, we have integrated all the above discussed algorithms into a single code file.

5. CHALLENGES FACED AND THE SOLUTION APPLIED

5.1 Challenges and Solutions

1. Mounting of motors at wheel joint. As we were using BO motors wheels, we weren't able to use for our N20 motors. The fix was we designed a mount in Solidworks and 3D printed that mount.

2. In the Original Design of our Bot, Gears were used.
   But then Gears were removed from the design, considering much torque was lost due to the gear mechanism. Instead, MG90s servos were used.

3. Length of legs and distance from the AOS were reconsidered many times.

4. Base plate of the bot had some design flaws regarding placement of holes for Servos and Mpu. Currently we have made holes manually by Drilling and the changes has been made in CAD Design.

5. No holes made in Base plate to for Castor Wheels (castor wheel used for experimental purposes like checking torque and rpm of motors in forward and backward direction). Currently we have made holes manually by Drilling and the changes has been made in CAD Design.

6. N20 heating up within a few moments of testing. As we were giving 12V directly to 6V N20's due to which they were heating

up. To fix this issue ,we used a LM2596 buck converter to convert the 12V to 6V.

7. During a few tests of Jumping, we found that the MG90 Servo gears either got broken or got misaligned due to high strain (sudden Jerk) on it during jumping. To fix this issue, we opened the Servos and tried fixing it and if the gear was completely broken we had to replace it with spare Mg90 servos or in worst case used Sg90 servos until the workaround is found or to perform some other tasks.

6. APPLICATIONS

6.1 MULTI TERRAIN ROBOTS

Multi Terrain RoBot which can change the length of individual legs based upon terrain and balance itself accordingly

6.2 OBSTACLE AVOIDANCE ROBOT

For obstacle avoidance ,we will use ultrasonic sensors (hc sr04) or Lidar sensors or IR sensors along with our present bot .
Lidar sensor has ability to measure 3d structures accurately .It uses laser beam.Cost is high.
Ultrasonic sensors use sound waves for detection. It's cost effective. Less sensor accuracy and limited detection range

Sensor fusion will enables us to improve obstacle detection and localization of objects in dark and foggy areas and also more accurate estimation of environment (wind speed etc)

Also for obstacle detection we can use computer vision also. (https://www.science.gov/topicpages/o/obstacle+detection+algorithm)

6.3 ADVANTAGES OF SELF-BALANCING ROBOTS

A two wheeled self balancing bot can cut quite a lot of the cost in designing the bot. You would require only two wheels, two sensors and a couple of the bot's body parts and you are ready to go. Also, a two wheeled bot has much more accuracy in terms of velocity and during turns than a three or four wheeled robot. We can have more control over a two wheeled robot as compared to other robots.

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion and Things Learned

To conclude, we would like to say that over the course of the past few weeks working on this project, we have learnt a lot of concepts, lessons and skills not only related to the project but also related to engineering and life in general. We had to do a lot of research and hard work because choosing to work in Control Systems and Designing our own BOT in the first place was like firing an arrow in the dark. We didn't know anything about Designing nor did we know whether we would be able to make such significant progress in this project or not. But thanks to the support and guidance of our mentors and seniors, we were able to make some significant progress in this project. We also learned how to research new topics, how to debug efficiently, how to write algorithms, how to work professionally on a project and much more.

What did we learn from the project?

1. We used SolidWorks in our project to design our robot. So we learned quite a bit about SolidWorks and designing.

2. We also used Autocad for rescaling and resizing 2D drawings which were given for Laser Cutting . This gave us the opportunity to understand a bit about these softwares.

3. We also learned about Hardware and how to handle them.

4. We understood what PID is, studied about PID controllers, how to implement PID, the uses and importance of PID, etc.

5. We also learned about algorithms of self-balancing .
We understood how a bot actually balances itself using readings from sensors like imu.

6. We also figured out the algorithm of Jumping.
We also learned about Servo motor accurate rotation capabilities on rotating them properly that can lead the bot to jump.

7.2 <u>Future aspects of Project</u>

1. Multi Terrain RoBot (changes length of legs based upon terrain and balance itself)

2. Implementing Positional control using LQR and roll stabilization.

3. Segway Bot (Assistant Type Robot)

4. For obstacle avoiding (using Ultrasonic sensors or Lidar and Computer Vision)

8. REFERENCES

8.1 Useful Links and References

Link of Our Github Repository

1. ESP-IDF Docs

2. Solidworks Playlist

3. Research Paper for Reference (MIT Paper)

4. Research Paper for Reference(Czech TU)

5. Reference Paper for PID