Self-Balancing Robot using Arduino

**Group P**

* Objective: -

Building and testing of a self-balancing robot using standard Arduino kit and Gyroscope.

* Tools & Electronics: -

1. Arduino Uno
2. MPU6050-Gyro/Accel
3. L293D Motor Driver
4. Breadboard
5. Gear Motors (150rpm) – 2
6. 9V batteries
7. Jumper wires
8. Acrylic sheet
9. Switches

* Difficulties Faced: -

1. Advanced concepts: -  
   The output the gyroscope provides needs an algorithm which processes the output data and causes some reaction based on it. The algorithm which is to be used is known as the PID algorithm. The problem is that the algorithm is time taking to understand and it has diverse applications, so to fit our purpose we had difficulty in applying it.
2. Adjusting constants: -

The algorithm requires that we set three constants, and these constants are to be set either by practical testing or by simulation. The simulation part is too advanced for us, so we are trying by trial and error method. So, even this took us a lot of time.

1. Libraries: -

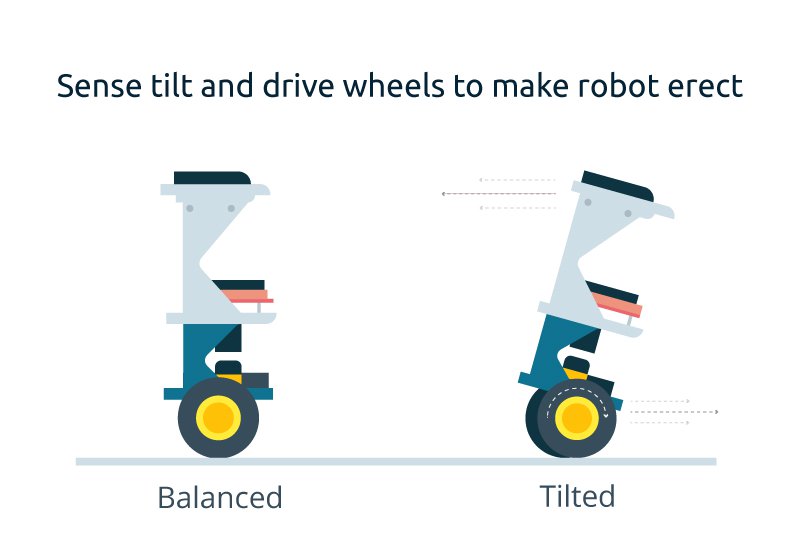
To implement the physical concepts, we needed to search through a lot of libraries to find a few containing the desired functions.

* Description and Solution: -

1. **Physics behind it: -**

Suppose the robot is tilted in towards the +ve X axis

Torque due to the weight would act in the -ve Z direction. So, to counteract this torque and balance the chassis, the wheels must rotate in the clockwise sense. This is the basic physics behind the solution.



1. **Implementation: -**

* We have used a gyroscope and an accelerometer module which would output voltage values which are proportional to the angular velocity and net acceleration respectively. These values contain information of how much we need to rotate the robot to reach the vertical state.
* Now, after having the values we implement the algorithm. Description: -

The PID (Proportional Integral Derivative) algorithm is an algorithm used to minimize an error value. In our case the error value is the difference between voltage of required position (set point) and the voltage corresponding to current position (Process variable).

e = Set Pt – Process Var

Now the algorithm requires three terms 1) proportional, 2) Integral and 3) Derivative.

1. Proportional: The value returned is product of error ‘e’ and the proportional constant Kp. Speed of response increases with increase in Kp while a large value of Kc would lead to violent oscillations.

P = e\*Kp

1. Integral: Reduces steady state error of the system (here, the robot).

I = sum of all errors

Iterate under loop:

I = error + I\*Ki

‘I’ here is initialized as the sum of all errors and then it is updated with existing information as well as new error output.

1. Derivative: Difference between current error and previous error.

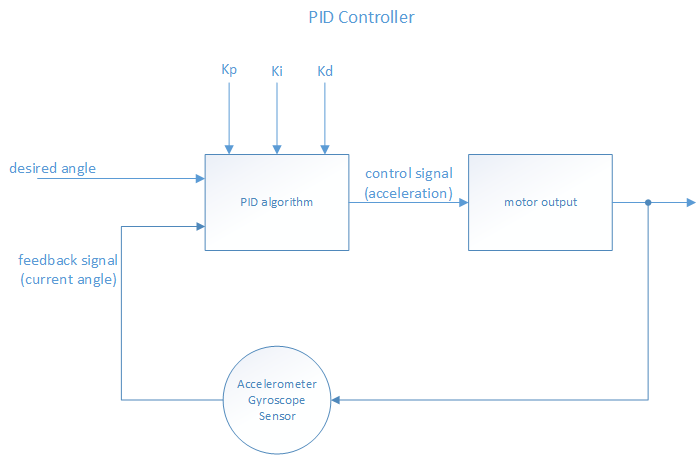
D = (current\_error – prev\_error)\*Kd

Now to get the total error we sum over the three terms. That is: -

Output = P + I + D

This value then rotates the motors in a specific direction and with a specific angular velocity all of which (Information) are contained inside the output value.

NOTE: *Kp, Ki and Kd must be repeatedly initialized on the basis of trial and error because only if the constants are correctly initialized we would get the required balancing effect and stability.*



* Individual Contribution: -

Members: -

* Devansh Singh Rathore (111701011)
* Nitin Vishwakarma (131701019)
* Ahmed Zaheer Dadarkar (111701002)
* Vipin Seth (1117010)
* Adarsh Kumar (101701003)
* Shwetank Verma (131701027)
* Vaibhav Jindal (111701029)
* Anish Jain (111701006)

Contributions: -

1. Devansh: -

* Source code analysis
* Connections analysis
* Chassis designing
* Testing

1. Nitin: -

* Source code analysis
* Made connections
* Connections analysis
* Chassis designing
* Testing

1. Vipin: -

* Wrote the code
* Code analysis
* Made Connections
* Connections analysis
* Chassis designing
* Testing

1. Ahmed: -

* Source code analysis
* Connections analysis
* Chassis designing
* Testing
* Report typed

1. Adarsh: -

* Source code analysis
* Connections analysis
* Chassis designing
* Testing

1. Shwetank: -

* Source code analysis
* Connections analysis
* Chassis designing
* Testing

1. Vaibhav: -

* Source code analysis
* Connections analysis
* Chassis designing
* Testing

1. Anish: -

* Source code analysis
* Connections analysis
* Chassis designing
* Testing
* Conclusion: -

We have successfully built a self-balancing robot based on principles of mechanics, electronics and statistical error analysis.

We have successfully tested the robot for corrections and verification.

We have learnt to use electronics for building products of consumer use.

We have learnt to apply algorithms of error analysis.