# Final Project - Self-Balancing Robot

## Final | Project Report


### Team Members

| | |
|---|---|
| Habiba Ramadan Mahmoud | 201700832 |
| Muhammad Younis El-Sawi | 201700915 |
| Riham Al-Sayed Abdo | 201601808 |
| Shaimaa Said Hassanen | 201700249 |

### Contact Info

s-mohammedyounis@zewailcity.edu.eg

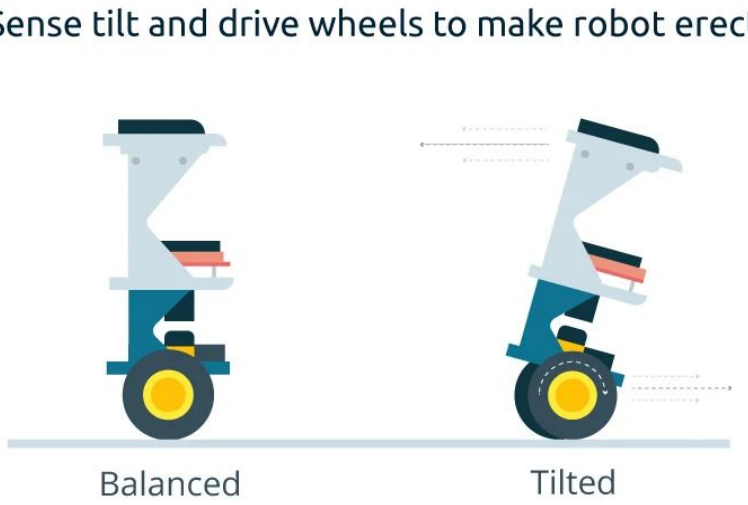

*Monday, January 25, 2021*

## Table of Contents

# **Introduction** | Self-Balancing Robot

## Introduction

Inverted pendulum applications are numerous; one example is the human body since it is an inverted pendulum balancing the upper body around the ankle joints in every step. Recently, Segways scooters have been using bodily movements for steering. What both applications have in common is that their center of mass is positioned above their pivot point, thus needing active control in order to balance. In the case of a self-balancing robot, it will fall either forward or backward. In order to keep the robot balanced, we need to keep its center of gravity above its pivot point. This can simply be achieved by driving its wheels in the direction in which it is falling.




This action's completion requires feedback and correcting elements. The feedback element will be a 6-axis accelerometer and gyroscope sensor combined, which gives both acceleration and rotation in all three axes. The Arduino uses this to calculate the current tilt angle. The correcting element shall be the motor and wheel combination. Moreover, a controller needs to be implemented to compensate for the said tilt. A PID-controller is able to control the pendulum angle since it is a SISO [Single-Input Single-Output] system. The PID controller's mission is to reduce the error to the smallest value possible by continually adjusting the output. The desired tilt in degrees will be calculated using software then used as an input. The 6-axis accelerometer and gyroscope sensor reads the current tilt of the robot, then feeds it to the PID algorithm, which performs calculations to control the motors and keep the robot in the upright position.

In this project, we are building a two-wheeled robot with two servo-motors connected to one wheel and controlled by an H-bridge. We are going to trigger the robot moving forward and backward using Bluetooth. The circuit diagram is shown below in fig(1).

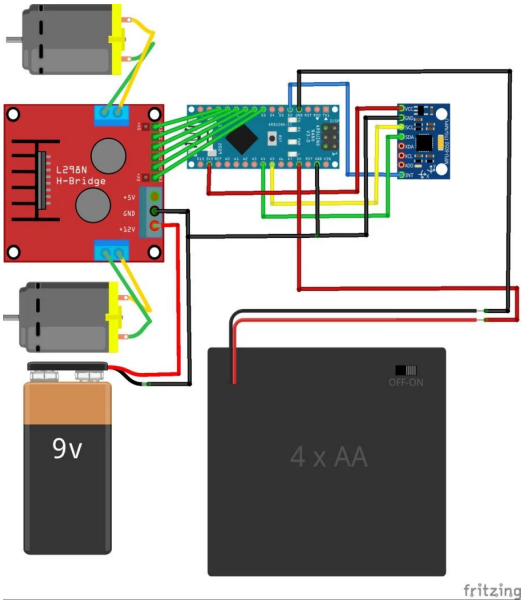


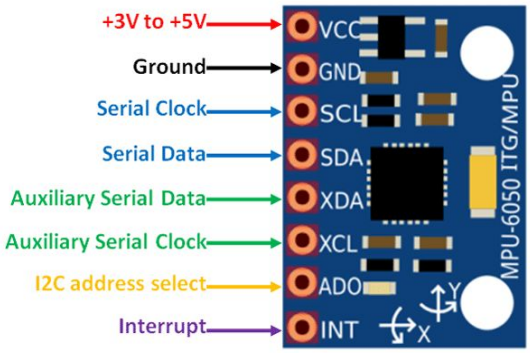

**Fig.1** Circuit diagram [1].

## Sensors

### **Gyroscope** | MPU 6050 6-axis

The accelerometer is used to measure the acceleration of moving objects over the three different referencing axes x, y and z. They can also be used to determine rotational motion by taking into account the gravitational acceleration of the earth. Earth gravity, movement or tilting action can cause forces of acceleration. Speed and displacement can also be calculated when the acceleration is multiplied by time. An accelerometer is a kind of a device-dependent on MEMS (Micro Electro Mechanical System) formed by a circuit having silicon seismic mass that variates its location as indicated by the orientation. Such accelerations on the three axes (x,y,z) are measured in terms of gravitational force $m/s^2$

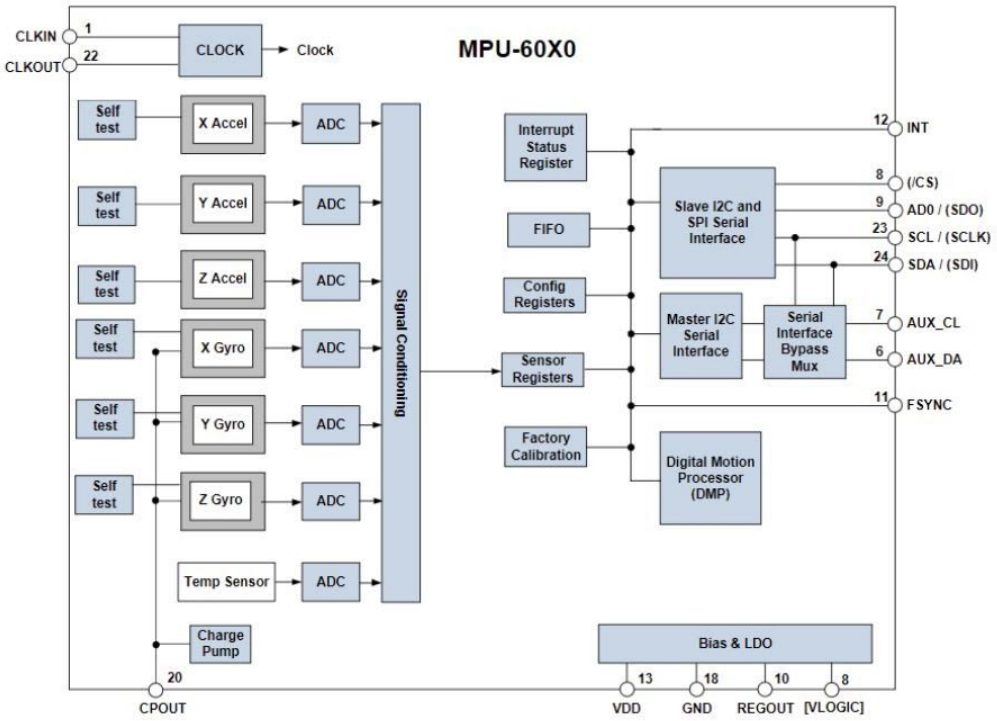
A gyroscope is used for measuring or maintaining orientation, that uses Earth's gravity and the principle of angular momentum. The structure of a gyroscope is like a turning wheel or disk in which the axle is allowed to expect any direction. Its structure comprises an openly turning plate called a rotor, mounted onto a turning hub in the focal point of a bigger and increasingly stable wheel. As the pivot turns, the rotor stays stationary to demonstrate the central gravitational force. [2]



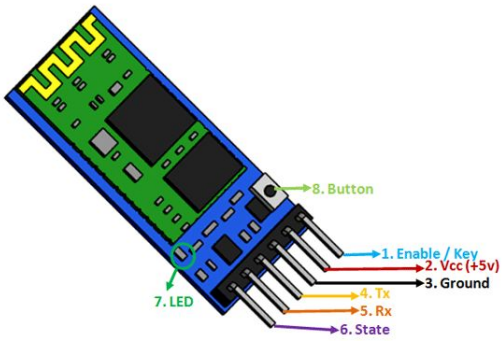

The accelerometer and gyroscope sensor shown above has the following input and output pins;

- ➔ VCC: To provide power to he sensor
- ➔ GND: is connected to the ground of the system.
- ➔ SCL: The serial clock used to provide the clock pulse for I2C communication to send the readings taken by the sensor to the microcontroller.
- ➔ SDA: The serial data line is what holds the data to be transmitted using I2C communication protocol.
- ➔ XDA and XCL Are auxiliary serial data and serial clock lines that can be used to interface other I2C modules with MPU 6050.
- ➔ AD0: I2C address select is used when more than one MPU is used on the same microcontroller to differentiate by using variant addresses.
- ➔ INT: interrupt pin is used to indicate when there is available data to read.

The MPU 6050 chip has the architecture shown in the figure below. It has a 3-axis gyroscope along with a 3-axis accelerometer. And 6 ADC are there to digitize the readings of each axis. One more ADC is there to digitize the reading of the temperature sensor but this is available only in some models which ours is not one of them. A total of seven 16 bit ADCs are shown in the figure below. The device also has a user-programmable gyroscope full-scale range of ±250, ±500, ±1000 and ±2000°/sec (dps) and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g and ±16g for precision tracking of both fast and slow motions.



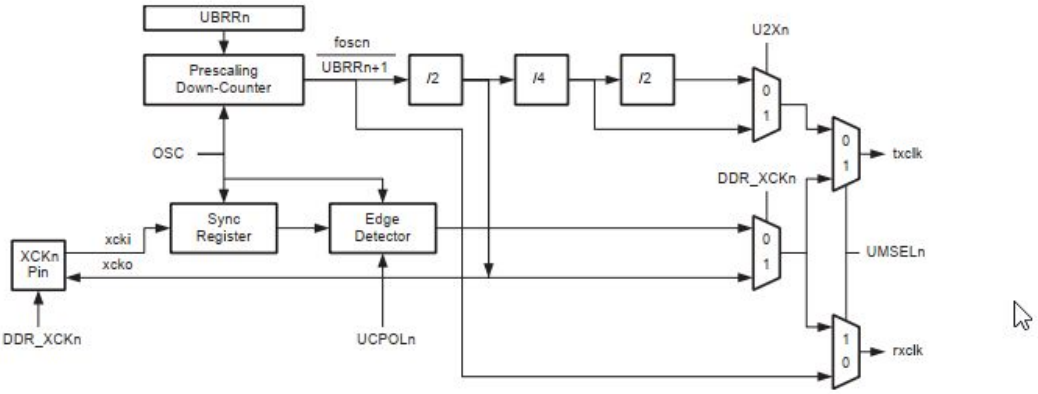

## **Bluetooth** | HC-05 Bluetooth module




The Bluetooth module is used to connect the robot with a mobile device where we can give orders to the robot to do. The pins VCC and Ground are used to power the module on. Tx and Rx are the transmitting and receiving serial data lines. Arduino pins connected to those pins are the corresponding Rx and Tx will transmit and receive data over bluetooth to the connected device (mobile phone in this case). State pin is used to connect to an on board LED checking that the module is working properly. The Button is used to switch between modes of the Bluetooth module, either Data mode or Command mode.The Enable/key pin works similarly to the button. By setting it low, we toggle the module to work on Data Mode, and high for the Command mode. The Data mode is where the device is used to transmit and receive data. The Command mode is where we need to change the default settings of the device.

## USART Programming

The three main components that we will need to instantiate before any communication between the bluetooth module and the arduino are the clock generator, the transmitter and the receiver. The initialization consists of setting the baud rate, setting frame format, and enabling the transmitter or the receiver.




1. The baud rate is generated by the system clock using both a prescaler and the above clock circuit. The USART Baud Rate Register (UBRR0) controls the programmable down counter / Prescaler in order to create a particular clock signal. The down-counter, which is running at the system's clock, is loaded with the UBRR0 value each time the counter has counted down to zero and generates a clock pulse.

**Table 19-1. Equations for Calculating Baud Rate Register Setting**

| Operating Mode | Equation for Calculating Baud Rate[1] | Equation for Calculating UBRRn Value |
|---|---|---|
| Asynchronous normal mode (U2Xn = 0) | $BAUD = \frac{f_{OSC}}{16(UBRRn + 1)}$ | $UBRRn = \frac{f_{OSC}}{16BAUD} - 1$ |

The above formula is used to calculate the correct value of UBBR0. For Atmega328p, the system clock runs at 16MHz. And since we intend to establish communication at speed 9600bps, the value of UBBR0 should be = ((16MHz / 16*9600) -1).

2. Then, we set the Frame Format using the register [UCSR0C].

USART accepts all combinations of the following as valid frame formats:

- ➔ 1 start bit.
- ➔ 5, 6, 7, 8, or 9 data bits.
- ➔ Disapled, even or odd parity bit.
- ➔ 1 or 2 stop bits

3. Then, we enable the receiver using the UCSR0B register and read the UBR0 register for the data received.

### Polling Issue

In polling the CPU wastes valuable time monitoring the USART registers. This valuable time could be used in the execution of other instructions as in, it should be used to balance the robot. This problem is solved by interrupt based transmission. In Atmega328p, we can enable an USART receive complete interrupt signal using RXCIE register. When doing so, the CPU becomes busy looping the main loop and whenever an unread data is received in the USART buffer, a RX complete interrupt is thrown and the CPU serves it in the ISR. The CPU doesn't have to monitor the USART register bits to check the status of the reception.

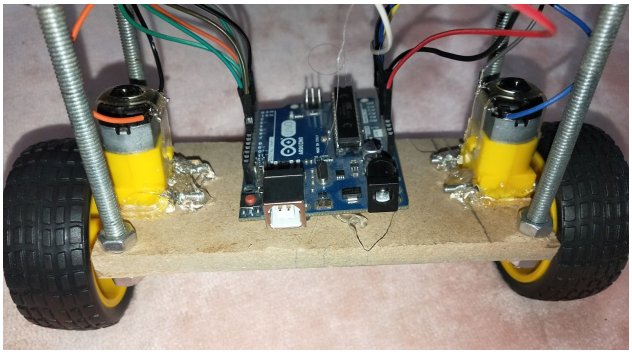
## Hardware Model

## **General Structure** | 3-Stage Model

Our model is composed of three stages with a total height of 18 cm. Theoretically, it becomes easier to balance a robot with a high altitude. So, we decided to make it a three-stage robot and to distribute the components evenly between the stages.

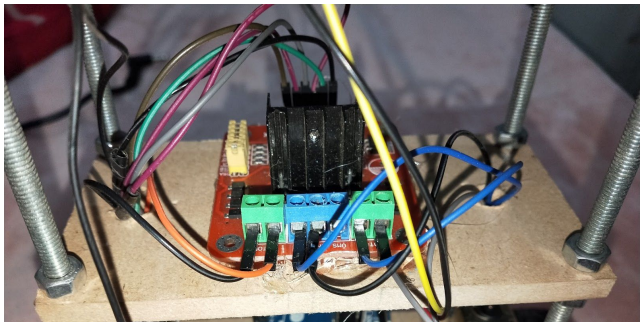
## **Stage 1** | Arduino and motors

The lower stage includes the motors, wheels and the microcontroller (Arduino UNO). the connected pins of the Arduino are digital 5, 6, 7, 8, 9, 10 for ENA, IN1, IN2, IN3, IN4, ENB respectively, digital pin 2 for IMU interrupt, Vin, GND for power and A4, A5 for I2C communication with the IMU.



## **Stage 2** | Motor driver

The middle stage contains the motor driver (H-Bridge) to control the DC motors using the microcontroller. The two motors are connected to the H-Bridge and it is responsible for providing power for the motors and the microcontroller from the 11.1 V battery.

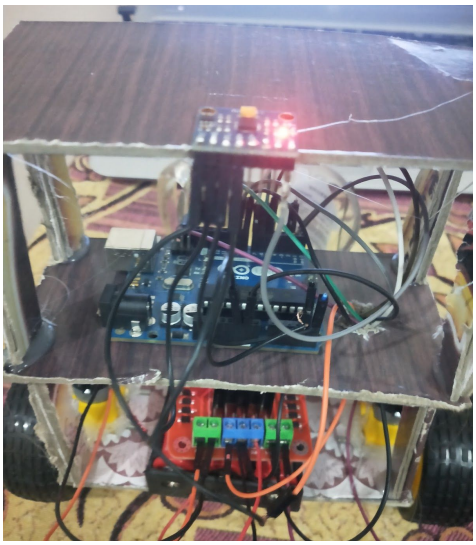


## **Stage 3** | IMU and Batteries

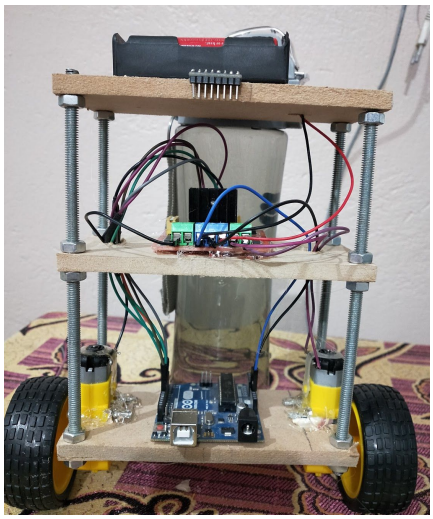
The upper stage contains the batteries and Gyroscope sensor (6-axis MPU 6050). The battery delivers power to the motor driver that power the motors and the microcontroller, the IMU senses the tilt angle and sends it to the microcontroller (Using I2C) to be used as feedback for the PID controller that generates the appropriate direction and speed for the motors to balance the robot.


## **Final Structure** | Wooden and Cardboard

We first started by constructing a model from Cardboard and it is shown in the figure below. The purpose of it was to check if the balancing would work or not and to estimate initial values for the PID parameters. The robot was balanced but with some minor problems in the performance like it was oscillating back and forth very quickly and sometimes it falls or hits a wall. These problems were mainly because the robot was very light from above so it oscillates easily.



As a final model, we re-constructed a model from fiber wood to be light in weight and strong to hold the components. To overcome the problems of the cardboard model, we moved the battery to the upper stage to provide some heaviness to help the robot to balance, as the self-balancing robot can be seen as an inverted pendulum so its center of mass needs to be higher than the wheels' axle to ease the process of balancing as the higher the center of mass, the lower angular acceleration which means slower fall.



## Control using PID

Self-balancing robot can be seen as an inverted pendulum, unlike the ordinary pendulum that tends to balance itself, it is impossible for the inverted pendulum to balance without external forces. To do so we need a controller, one of the most widely used controllers is the PID controller. Simply, it takes some error as input and in response it generates an appropriate output for correction. In our case, we want to control the tilt angle of the robot, such that when the robot deviates from the desired set point (ideally it is 180) the PID controller sends a command to the wheels to move in a certain direction with a certain speed to restore the robot to the setpoint. The output of the PID controller is calculated based on the equations shown in the figure below, where Kp, Ki and Kd are the controller parameters.

$$Output = Kp.e(t) + Ki\int e(t).dt + Kd.\frac{d}{dt}e(t)$$

where $e(t) = setpoint - input$

$$\begin{aligned} Output &= Kp * (error) + Ki * (errorSum) * dt + Kd * (currentError - previousError)/dt \\ &= Kp * (error) + Ki * (errorSum) * sampleTime - Kd * (currentAngle - previousAngle)/sampleTime \end{aligned}$$

## The Desired Position | Setpoint

Theoretically speaking, the ideal setpoint is 180 degrees and this happens if we have a perfect center of gravity and the model is built perfectly. Usually, it is not the case, but the practical setpoint in a good model is not very far from the ideal value. In our case, it was 178 degrees.

## Proportional Gain | Kp

The proportional gain, from its name, is responsible for generating an output proportional to the error, so if the robot has slightly deviated from the desired position, the output would be a very small speed for the motors.

## Integral Gain | Ki

The integral gain generates an output based on the sum of the previous errors, so it considers the system performance in the past to generate an appropriate output.

## Differential Gain | Kd

The differential gain accommodates for the future performance of the robot, as it generates the output based on the current error and the previous error so it is a predictive term that considers the performance at the upcoming sample.

## PID Tuning | Kp=50, Ki=70, Kd=1.1

PID tuning is the most time-consuming stage as it is very crucial for successful balancing. There are many ways in control theory for PID tuning including software methods like MATLAB and Simulink. However, in our case, the manual tuning was enough. We started by setting all the parameters to 0, then we started increasing the value of Kp. At small values (<35) the robot has a very slow response and it cannot recover its position, at very large values (>80) the robot oscillates very widely and then falls over the ground, at the appropriate value (50) it can just balance with some human help. Once Kp was set, we started tuning Kd. Kd can lessen the oscillations and the overshoot greatly, at the appropriate value (1.1) the robot can stand even if it is pushed. The final step is to tune Ki which is responsible for the rising time, so at the suitable value (70) the robot can balance very quickly without wild oscillations.

## References

[1] M. Elamrani, "How to Build an Arduino Self-Balancing Robot | Arduino", Maker Pro, 2017. [Online]. Available: https://maker.pro/arduino/projects/build-arduino-self-balancing-robot. [Accessed: 11- Nov- 2020].

[2] Kirbas, I, “Developing a signal similarity analyse software for accelerometer sensor data”, Journal of International Scientific Publications, 2019.

[3] S. midhun, "Arduino Self-Balancing Robot", Instructables.com, 2017. [Online]. Available: https://www.instructables.com/Arduino-Self-Balancing-Robot-1/. [Accessed: 24- Jan- 2021].

[4] A. Raj, "DIY Self Balancing Robot using Arduino", Circuit Digest, 2018. [Online]. Available: https://circuitdigest.com/microcontroller-projects/arduino-based-self-balancing-robot. [Accessed: 24- Jan- 2021].