Mechatronics Project: Two Wheeled Self-Balancing Robot

Technical Report · May 2023

DOI: 10.13140/RG.2.2.10832.48640

6 authors, including:

Osama Halawa
Helwan University
4 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Alber Maged
Helwan University
4 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Eslam Tarik
Helwan University
5 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Helwan University

Faculty of Engineering at El-Mataria

Mechanical Power Department

Mechatronics Project: Self-Balance Robot

Prepared by:

Section (2) Team (1)

1- Osama Mamdouh Mostafa
2- Alber Maged Kamal
3- Andrew George Sabry
4- Ayatallah Ramadan Mahmoud
5- Ahmed Mohamed Gaber
6- Eslam Tarik Saad

Under supervision of

Assoc. Prof. Osama Ismail

Eng. Beshoy

14/5/2023

Abstract

This project aims to design and implement a self-balancing robot using an Arduino UNO microcontroller, MPU6050 accelerometer and gyroscope sensor, and L298N motor driver. The robot utilizes a PID control algorithm to balance itself on two wheels.

The report provides an overview of the theory of operation, project components, and block diagram, as well as detailed descriptions of the MPU6050 sensor and L298N motor driver. Additionally, the report outlines the process of PID tuning and provides suggestions for future improvements, such as adding line-following capabilities or upgrading to higher-quality components. The self-balancing robot demonstrates the potential of using PID control algorithms to stabilize dynamic systems and could have potential applications in the fields of robotics and automation.

Table of Contents

List of Figures

List of tables

1 Theory of Operation

Self-balancing robot is based on the principle of inverted pendulum, which is a two-wheel vehicle balances itself up in the vertical position with reference to the ground. It consists of both hardware and software implementation.

1.1 Operation Principle

Due to the irregular distribution of masses of the robot, the center of mass is displaced from the vertical position (y-axis) thus the robot falls.

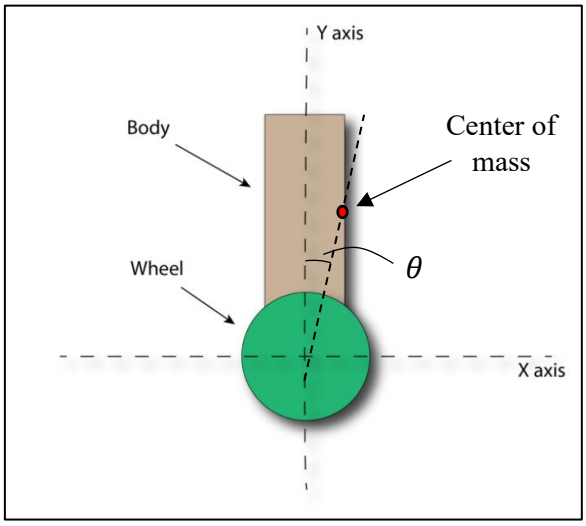

Fig. 1.1: Inclination angle of a self-balance robot

If the robot tilts forward or backward, moving the wheels in the same direction can restore the stability of the robot by producing a torque that counteracts the tilt. The speed of the wheels will depend on the degree of tilt.

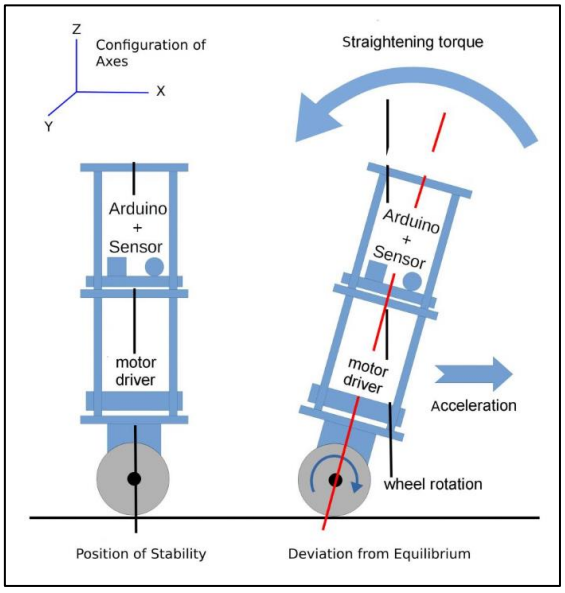

Fig. 1.2: Principle of self-balancing

1.2 Applications of Self-balancing robots

1.2.1 Hoverboards (Segways)

It contains 2 gyroscopes and 2 motors (one in each wheel) so when the both foot tilt forward it move forward, when both foot tilt backward it move backward and when one foot tilt backward and the other forward it spins. The motion will not stop until the foot returns to reference position (horizontal).

Fig. 1.4: Hoverboard

Fig. 1.3: Segway

1.2.2 Ascento: The Two-Wheeled Jumping Robot

It is a small agile jumping robot designed to transverse all-terrain environments. It has a combination of springs and high torque motors that enables it to move quickly on flat terrain and can balance and jump over obstacles.

Fig. 1.5: Ascento: The Two-Wheeled Jumping Robot

2 Project Components

We used various components in our project. The components were selected based on our needs, availability, and cost.

2.1 Chassis

Our robot chassis is 3D printed from PLA material, designed to have a full height of 17cm and it contains three levels that hold the components of the robot.

The first level which contains the motor driver (L298N).

The second level contains the battery holder.

The third level contains the Arduino board (UNO) and the gyroscope sensor (MPU6050).

All the levels are designed to fit the width and dimensions of its components.

The second and third levels have slots for the purpose of wire management.

There are four holes to install the motor mounts which hold down the geared motor and the wheels in a position as close as possible to the center of gravity of the robot.

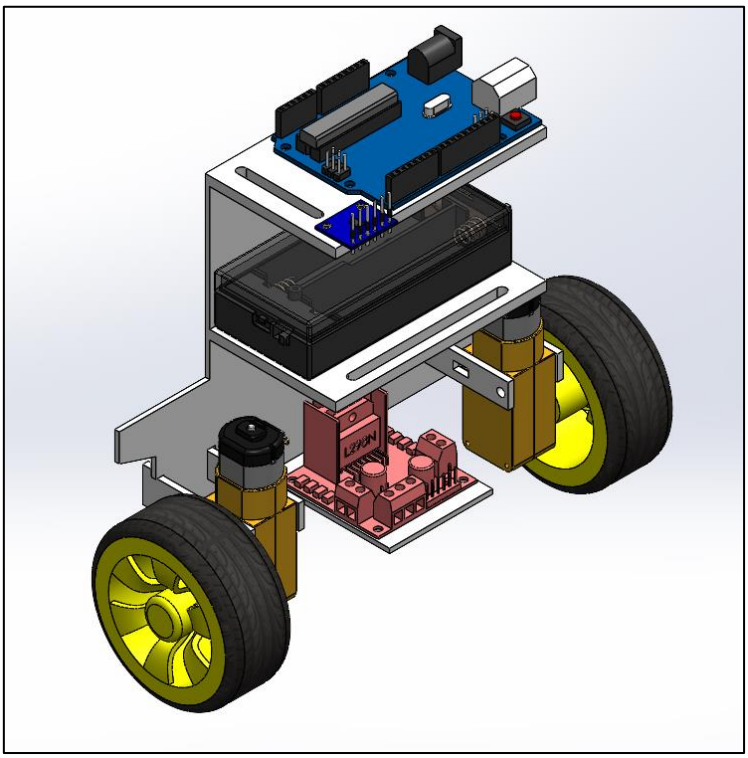

Fig. 2.1: Assembly of the project

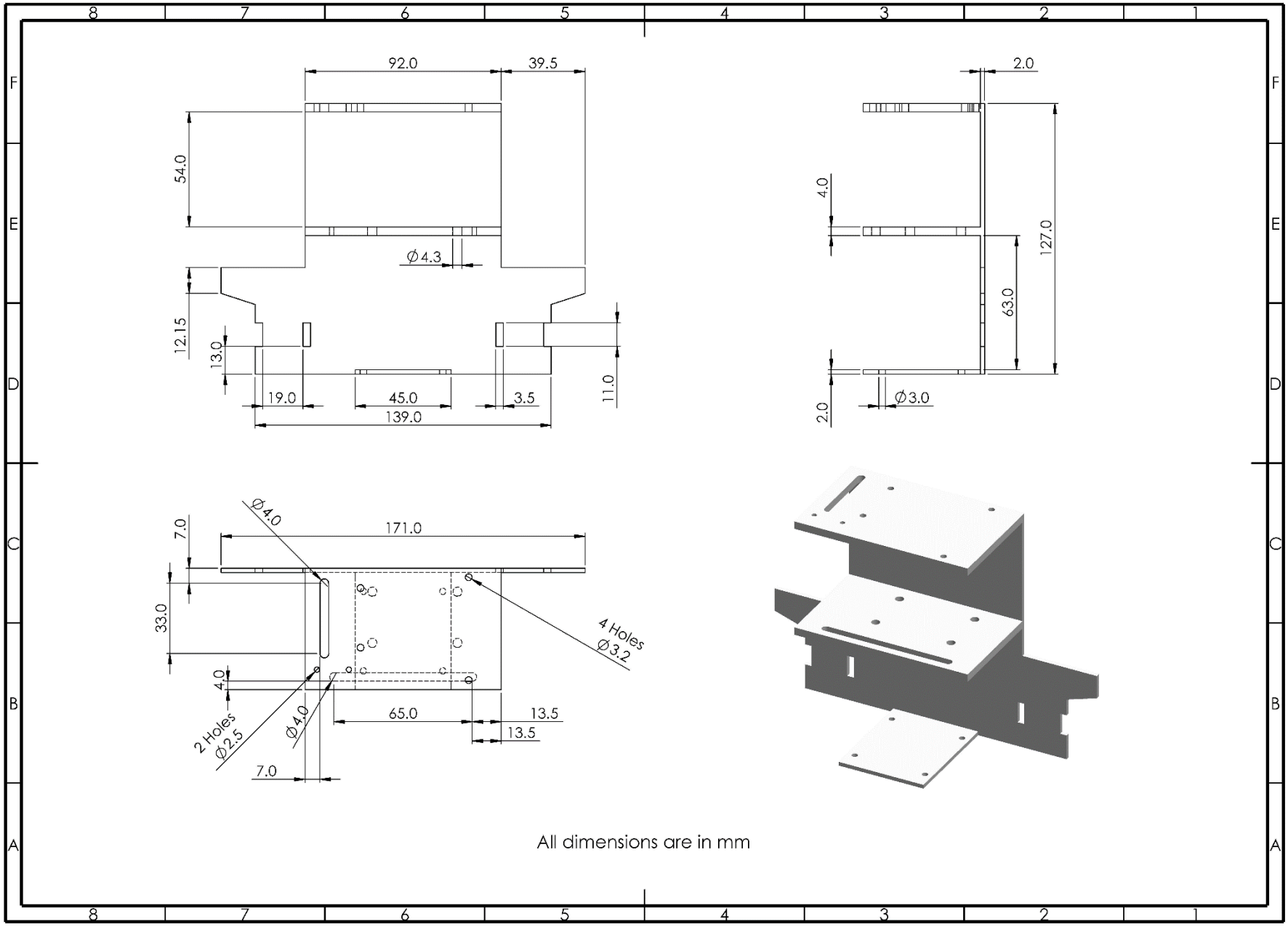

Fig. 2.2: Chassis working drawing

2.2 Two Yellow DC Geared Motors

in our project we used 2 of yellow DC Geared motors with these Specifications:

Table 2.1: DC motor specifications

Operating Voltage	3:12 V
Shaft Length	8.5 mm
Shaft Diameter	5.5 mm
No Load Current	40:180 mA
Rated Speed After Reduction	100 RPM
Rated Torque	0.35 kg.cm
Weight	30 gm
Gearbox Shape	Straight
Dimensions (LxWxH)	(70x23x19) mm

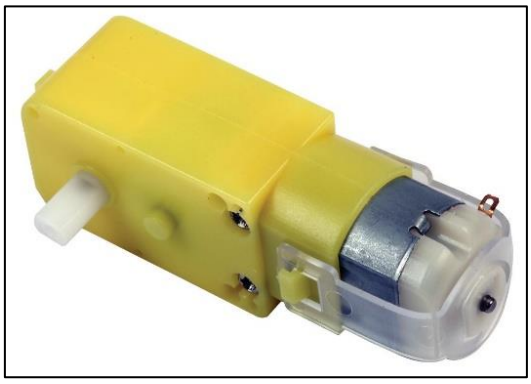

Fig. 2.3: Yellow DC Geared Motor

❖ We choose this motor because:

1. Low density: lightweight, low inertia.
2. Motor gives good torque and rpm at lower operating voltages.
3. A small shaft with matching wheels gives an optimized design of robot.
4. Capability to absorb shock and vibration because of elastic compliance.
5. Ability to operate with minimum or no lubrication, due to inherent lubricity.
6. The relatively low coefficient of friction.

2.3 MPU6050 (accelerometer & gyroscope sensor)

The MPU6050 is a Micro-Electro-Mechanical Systems (MEMS) that consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motion-related parameters of a system or object. This module also has a (DMP) Digital Motion Processor inside it which is powerful enough to perform complex calculations and thus free up the work for Microcontroller.

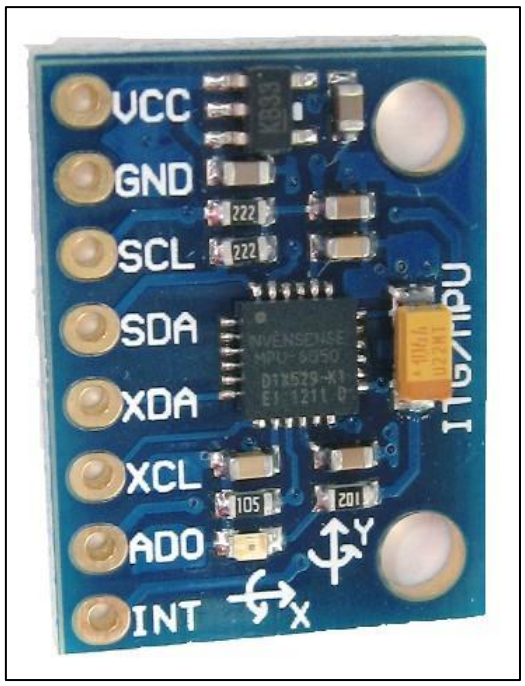

Fig. 2.4: MPU6050 Sensor

The module also has two auxiliary pins which can be used to interface external IIC modules like a magnetometer, however, it is optional. Since the IIC address of the module is configurable more than one MPU6050 sensor can be interfaced to a Microcontroller using the AD0 pin. This module also has well documented and revised libraries available hence it's very easy to use with famous platforms like Arduino.

2.3.1 Five steps to choose our sensor

1. Signal Nature (Physical property): We need to measure angle of tilt or inclination.

2. Output Nature: Electrical signal (mv) that then by to motor convert to torque.

3. Measuring range: We need measure angle of inclination in range between -90° to 90°

4. Measuring characteristic:
   - Sensitivity of Accelerometer: (16384, 8192, 4096, or 2048) [LSB/sec]
   - Sensitivity of Gyroscope: (131, 65.5, 32.8, or 16.4) [LSB/(°/s)]

5. Cost & Availability: 110 EGP and available in any hardware store.

2.3.2 Accelerometer working principle

MEMS accelerometers are used wherever there is a need to measure linear motion, either movement, shock, or vibration but without a fixed reference. They measure the linear acceleration of whatever they are attached to. All accelerometers work on the principle of a mass on a spring, when the thing they are attached to accelerates, then the mass wants to remain stationary due to its inertia and therefore the spring is stretched or compressed, creating a force which is detected and corresponds to the applied acceleration.

When the sensor is subjected to a linear acceleration along its sensitive axis, the proof mass tends to resist motion due to its inertia, therefore the mass and its fingers become displaced concerning the fixed electrode fingers. The gas between the fingers provides a damping effect. This displacement induces a differential capacitance between the moving and fixed silicon fingers which is proportional to the applied acceleration. This change in capacitance is measured with a high-resolution ADC and then the acceleration is calculated from the rate of change in capacitance. In MPU6050 this is then converted into readable value and then it's transferred to the I2C master device.

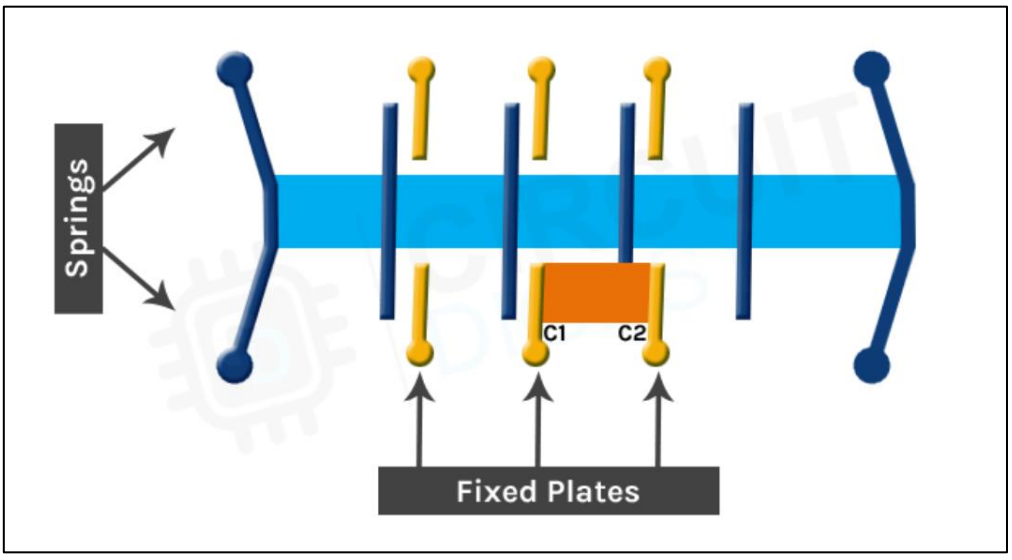

Fig. 2.5: Accelerometer working principle

2.3.3 Gyroscope working principle

The MEMS Gyroscope working is based on the Coriolis Effect. The Coriolis Effect states that when a mass moves in a particular direction with velocity and an external angular motion is applied to it, a force is generated and that causes a perpendicular displacement of the mass. The force that is generated is called the Coriolis Force and this phenomenon is known as the Coriolis Effect. The rate of displacement will be directly related to the angular motion applied.

The MEMS Gyroscope contains a set of four proof masses and is kept in a continuous oscillating movement. When an angular motion is applied, the Coriolis Effect causes a change in capacitance between the masses depending on the axis of the angular movement. This change in capacitance is sensed and then converted into a reading.

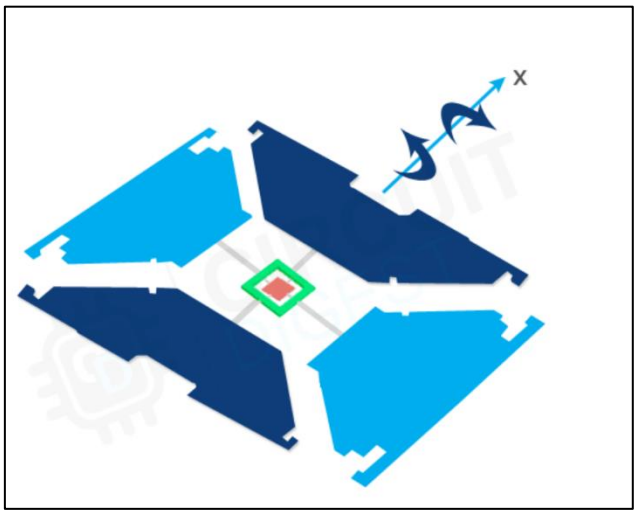

Fig. 2.6: Angular displacement causes change in capacitance

2.3.4 DMP (Digital Motion Processor)

The embedded Digital Motion Processor (DMP) is used to compute motion processing algorithms. It takes data from gyroscope, accelerometer, and additional 3rd party sensor, if connected, such as magnetometer and processes the data. It provides motion data like roll, pitch, yaw angles, landscape, and portrait sense etc. It minimizes the processes of host in computing motion data, the resulting data can be read from DMP registers.

2.3.5 MPU6050 pinouts

The MPU6050 module has a total of 8 pins. In which at least 4 pins are necessary for the interfacing. The pinout of a MPU6050 module is as follows:

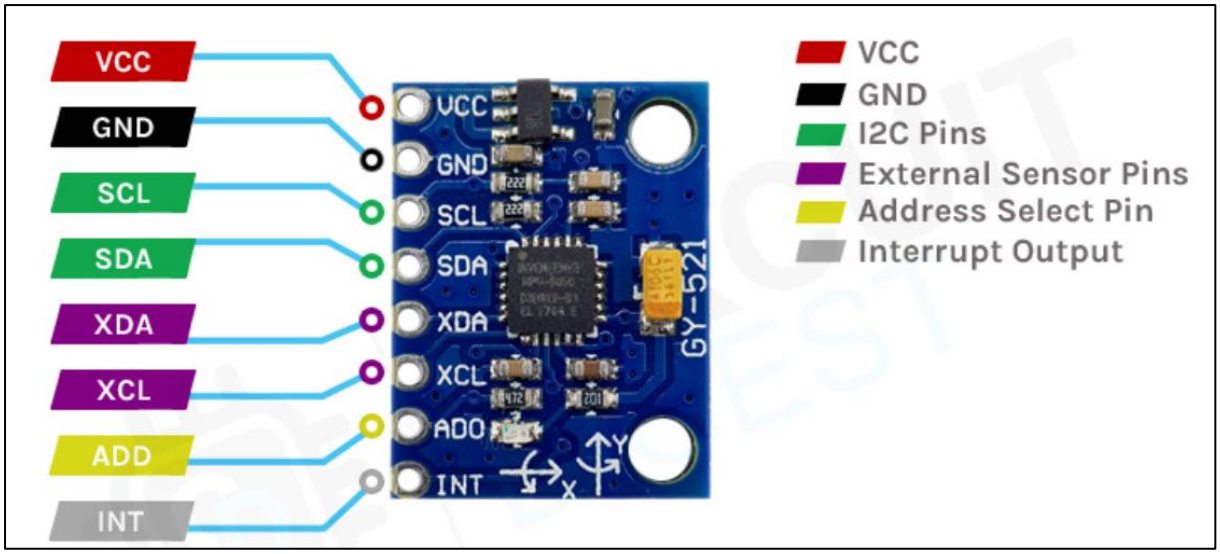

Fig. 2.7: MPU6050 Pinout

Table 2.2: MPU6050 Pinouts

VCC	Provides power for the module, Connect to the 5V pin of the Arduino.
GND	Ground Connected to Ground pin of the Arduino.
SCL	Serial Clock Used for providing clock pulse for I2C Communication.
SDA	Serial Data Used for transferring Data through I2C communication.
XDA	Auxiliary Serial Data - Can be used to interface other I2C modules with MPU6050.
XCL	Auxiliary Serial Clock - Can be used to interface other I2C modules with MPU6050.
ADO	Address select pin if multiple MPU6050 modules are used.
INT	Interrupt pin to indicate that data is available for MCU to read.

2.3.6 MPU6050 specifications

Table 2.3: MPU6050 Specifications

Operating Voltage	3:5 V
Gyro range ($\hat{A}°/s$)	$\hat{A} \pm 250, 500, 1000, 2000$
Acceleration range (g)	$\hat{A} \pm 2, \hat{A} \pm 4, \hat{A} \pm 8, \hat{A} \pm 16$
Length	20 mm
Width	16 mm
Hight	1 mm

2.4 L298N Motor Driver

L298N module is a high voltage, high current dual full-bridge motor driver module for controlling DC motor and stepper motor. It can control both the speed and rotation direction of two DC motors. This module consists of an L298 dual-channel H-Bridge motor driver IC. This module uses two techniques for the control speed and rotation direction of the DC motors. These are PWM (Pulse Width Modulation) for controlling the speed and H-Bridge for controlling rotation direction.

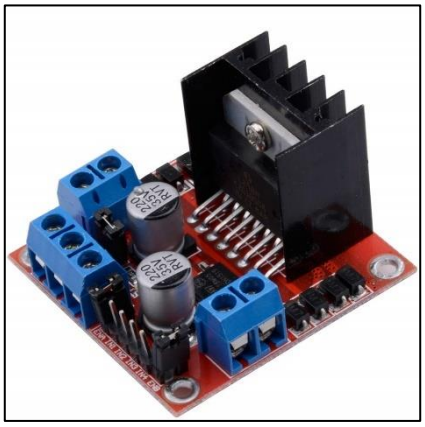

Fig. 2.8: L298N Module

2.4.1 PWM to control speed

The speed of a DC motor can be controlled by changing its input voltage. A widely used technique to accomplish this is Pulse Width Modulation (PWM).

PWM is a technique in which the average value of the input voltage is adjusted by sending a series of ON-OFF pulses. This average voltage is proportional to the width of the pulses, which is referred to as the Duty Cycle. The higher the duty cycle, the higher the average voltage applied to the DC motor, resulting in an increase in motor speed. The shorter the duty cycle, the lower the average voltage applied to the DC motor, resulting in a decrease in motor speed.

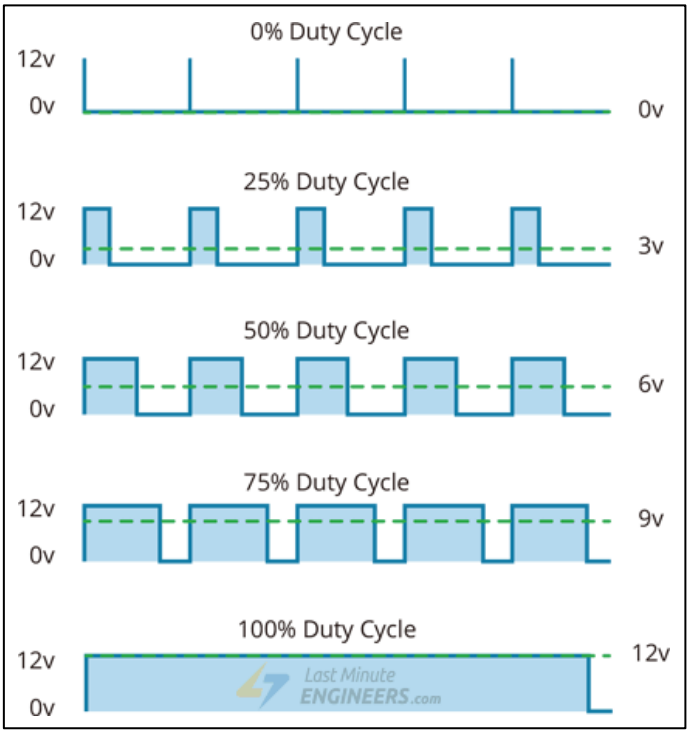

Fig. 2.9: PWM & Duty Cycle

2.4.2 H-Bridge to control the direction of rotation

The spinning direction of a DC motor can be controlled by changing the polarity of its input voltage. A widely used technique to accomplish this is to use an H-bridge. An H-bridge circuit is made up of four switches arranged in an H-shape, with the motor in the center. Closing two specific switches at the same time reverses the polarity of the voltage applied to the motor. This causes a change in the spinning direction of the motor.

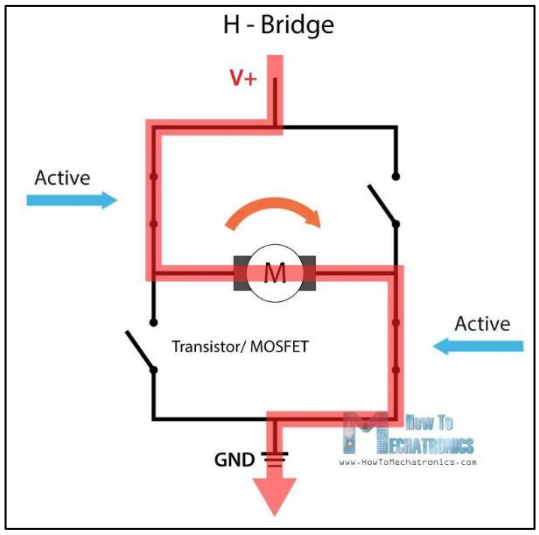

Fig. 2.10: H-bridge

2.4.3 L298N Motor Driver Module Pins:

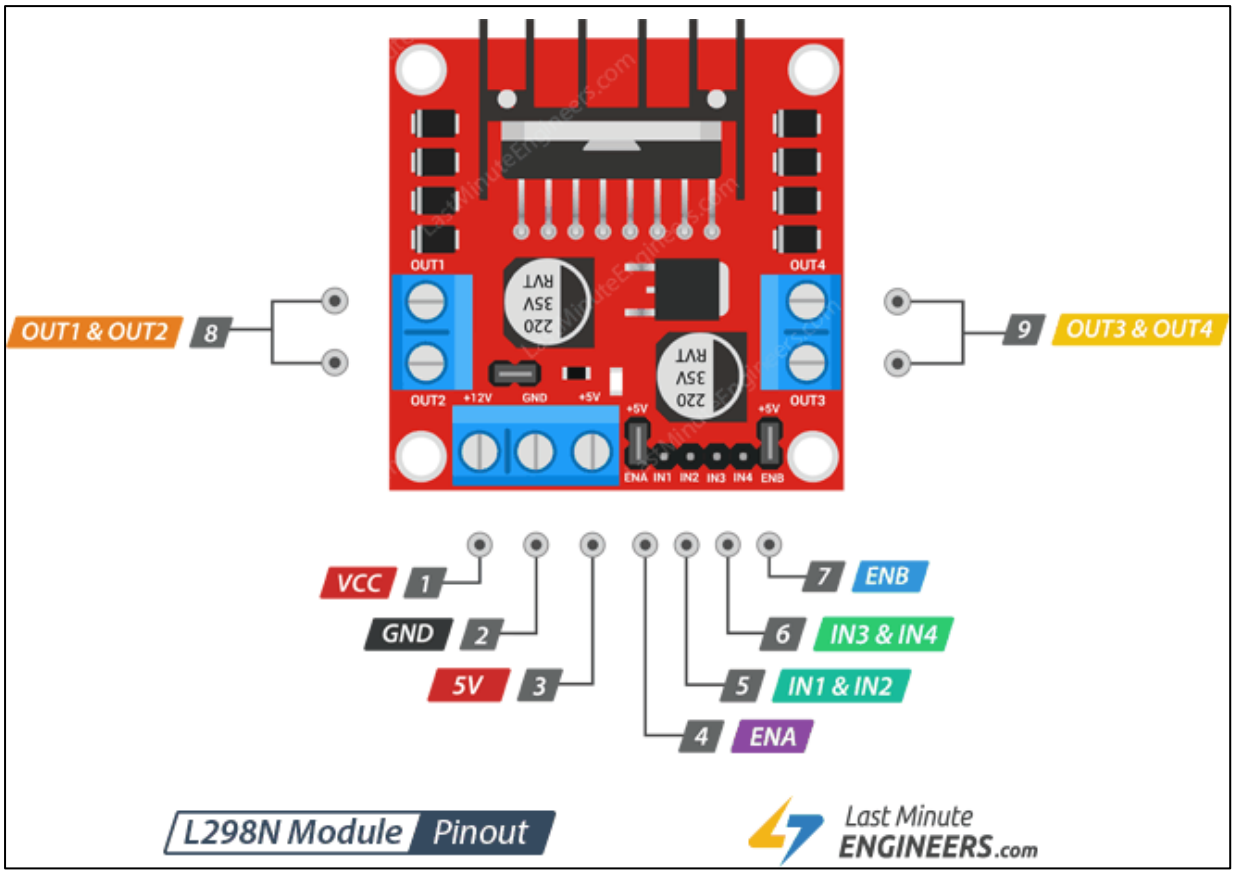

Fig. 2.11: L298N Module Pins

Table 2.4: L298N Pins description

Pin No.	Pin Name	Description
Power Supply Pins		
1	**VCC**	VCC pin is used to supply power to the motor. Its input voltage is between 5 to 35V
2	**GND**	GND is a ground pin. It needs to be connected to the power supply ground (negative).
3	**+5V**	+5V pin supplies power for the switching logic circuitry inside the L298N IC. If the 5V-EN jumper is in place, this pin acts as output and can be used to power up a microcontroller or other circuitry (sensor). If the 5V-EN jumper is removed, you need to connect it to the 5V power supply of the microcontroller.
Direction Control Pins		
5	**IN1**	These pins are input pins of **Motor A**. These are used to control the rotating direction of Motor A. When one of them is HIGH and the other is LOW, Motor A will start rotating in a particular direction. If both the inputs are either HIGH or LOW the Motor A will stop
	IN2	
6	**IN3**	These pins are input pins of **Motor B**. These are used to control the rotating direction of Motor A. When one of them is HIGH and the other is LOW, Motor A will start rotating in a particular direction. If both the inputs are either HIGH or LOW the Motor A will stop
	IN4	
Speed Control Pins		
4	**ENA**	ENA pin is used to control the speed of **Motor A**. If a jumper is present on this pin, so the pin is connected to +5V and the motor will be enabled, then the Motor A rotates maximum speed. If we remove the jumper, we need to connect this pin to a PWM input of the microcontroller. In that way, we can control the speed of Motor A. If we connect this pin to Ground the Motor A will be disabled.
7	**ENB**	ENB pin is used to control the speed of **Motor B**. If a jumper is present on this pin, the pin connected to +5V and the motor will be enabled, then the Motor B rotates maximum speed. If we remove the jumper, we need to connect this pin to a PWM input of the microcontroller. In that way, we can control the speed of Motor B. If we connect this pin to Ground the Motor B will be disabled

8	OUT1 & OUT2	This terminal block will provide the output for **Motor A**.
9	OUT3 & OUT4	This terminal block will provide the output for **Motor B**.

2.4.4 L298N Specifications

Table 2.5: L298N Specifications

Motor output voltage	5V : 35V
Motor output voltage (Recommended)	7V – 12V
Logic input voltage	5V – 7V
Continuous current per channel	2A
Max Power Dissipation	25W

2.5 Li-Ion Battery (18650 Blue)

An 18650 battery is a type of lithium-ion rechargeable battery. The numbers "18650" refer to the battery's dimensions: it is 18mm in diameter and 65mm in length. 18650 batteries are commonly used in electronic devices such as laptops and flashlights, as well as in electric vehicles and other high-power applications. They are known for their high energy density, long lifespan, and relatively low self-discharge rate. Multiple batteries can be connected in series to get higher voltage or in parallel to get higher capacity.

Fig. 2.12: 18650 Li-Ion Battery

One Li-Ion battery produces 3.7V and has a capacity of 2000mAh.

2.6 Battery Case

An 18650 Battery Case Holder 2 cells with on-off Switch is used to hold the batteries and to connect them in series so that the output voltage is 7.4V.

We branched the output wires so that we could connect Arduino UNO and L298N module with the same batteries.

Fig. 2.13: Two cell 18650 Batter Case

2.7 Arduino UNO

Arduino UNO is one of the most popular microcontrollers boards in the Arduino product line. We mainly choose Arduino UNO over the other types like Nano, Mega, Due, etc. because it satisfies the following:

- Simple and easy to use.
- It doesn't require a separate chip for USB connectivity.
- It can be powered using a battery or a DC power supply.
- Availability in the market.
- Inexpensive in comparison to other microcontrollers.
- It has the required number of analog and digital pins that are needed to complete the project.
- Small in size.

Fig. 2.14: Arduino UNO

2.8 Weights & Prices

The following table shows the weight and price of each component:

Table 2.6: Components Weights & Prices

Component	Quantity	Weight per unit (gm)	Total Price (EGP)
Chassis	1	132	264
Yellow DC Geared Motors	2	30	90
MPU6050	1	10	110
L298N Motor Driver	1	26	80
Arduino UNO	1	28	350
18650 Li-Ion battery	2	43	110
Battery Case	1	33	20
Wheels	2	28	50
Charger	1	-	100
Male-Male wires	20	3.3	20
Male-Female wires	20	3	20
Total	**-**	**476**	**1214**

We only used about 15 male-female wire that is why the total is only 476 grams. Also, we don't need the Charger's weight as it won't be in the assembly of the robot.

2.9 Connection Diagram

The following diagram shows the wiring between the components:

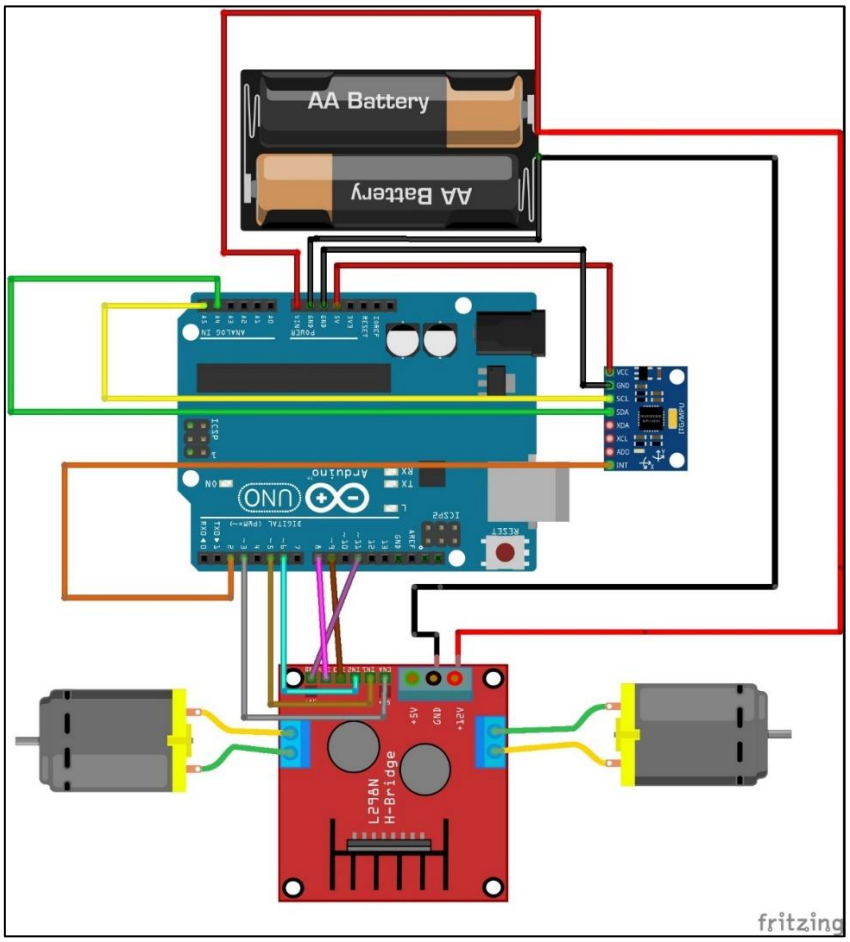

Fig. 2.15: Connection Diagram, this diagram was made using Fritzing software.

The following table also shows the wiring made between the components.

Table 2.7: Connections between components

From Pin	To pin
Power Supply	
Li-Ion battery (7.4V) +ve (RED)	V_{in} in Arduino
Li-Ion battery (7.4V) -ve (Black)	GND in Arduino
Li-Ion battery (7.4V) +ve (RED)	+12V in L298N
Li-Ion battery (7.4V) -ve (Black)	GND in L298N

MPU6050	
VCC	5V
GND	GND
SCL	A5
SDA	A4
INT	2
L298N	
ENA	3~
IN1	5~
IN2	6~
IN3	9~
IN4	10~
ENB	11~
DC Motors	
OUT1	Motor 1 closer to chassis wire
OUT2	Motor 1 far from chassis wire
OUT3	Motor 2 closer to chassis wire
OUT4	Motor 2 far from chassis wire

3 Block Diagram

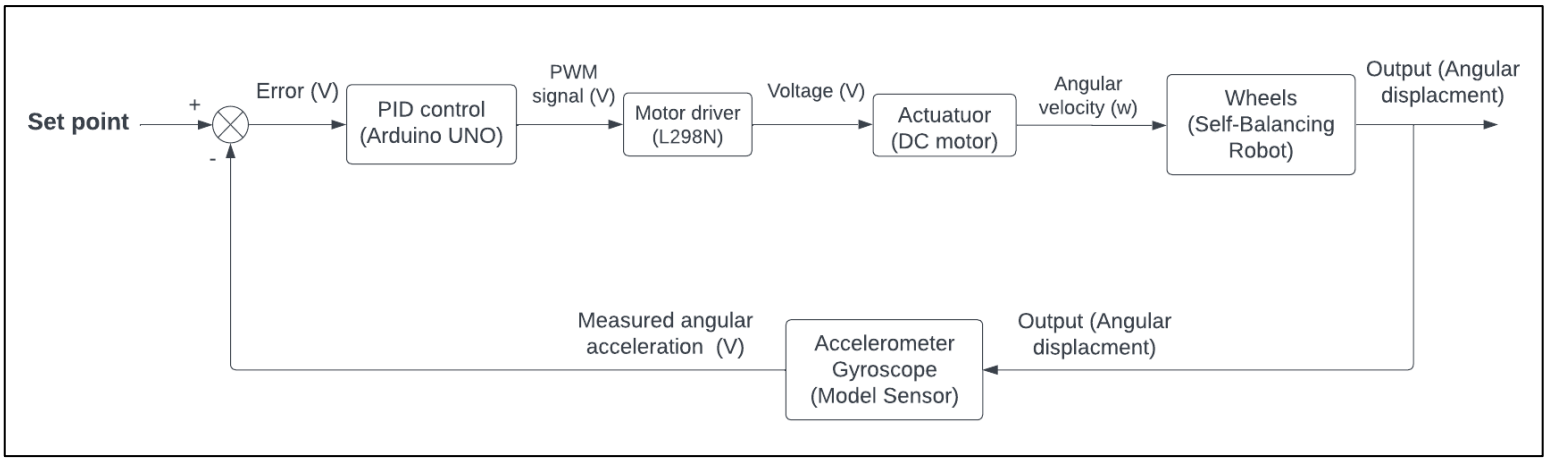

Fig. 3.1: Self-Balance Robot Block Diagram

3.1 The Process

As we can see from the illustrated block diagram, we can clarify the whole process of the self-balancing robot which is represented in the block diagram as the wheels because it is the main controlled part.

Arduino UNO (the controller) will compare the set point and the measured value from the sensor and produce an Error value, then it will send an actuating signal to the DC motor via L298N module so that the motors receive a specific direction and speed from the module which will turn the wheels and balance the robot. If any change done on the robot, the MPU6050 sensor will sense this change by sensing the output of the wheels and convert this output into voltage for Arduino UNO to compare. This is a closed-loop feedback control system where the corrective actions are done because of the feedback of the sensor and the comparison made by the controller (Arduino UNO).

3.2 Block Diagram Components

- The wheels which are the system we study
- Accelerometer gyroscope which is the sensor responsible for measuring angular acceleration
- Arduino UNO which is the microcontroller, and it is responsible for sending and receiving signals to motor drivers and MPU6050 sensor.
- L298N motor driver which is responsible for controlling DC motors as it can control both the speed and rotation direction of the two DC motors.
- DC motors represent the actuator and produce motion.

4 Codes

In our project we used two codes, one for calibrating the MPU6050 sensor and the other for the self-balance robot itself.

4.1 MPU6050 Calibration Code

This code is used to calibrate the MPU6050, we had to make our MPU6050 perfectly horizontal on a table and not touch it for the calibration to be correct.

Code:

```
// Arduino sketch that returns calibration offsets for MPU6050 //   Version
1.1  (31th January 2014)
// Done by Luis Ródenas <luisrodenaslorda@gmail.com>
// Based on the I2Cdev library and previous work by Jeff Rowberg
<jeff@rowberg.net>
// Updates (of the library) should (hopefully) always be available at
https://github.com/jrowberg/i2cdevlib

// These offsets were meant to calibrate MPU6050's internal DMP, but can be
also useful for reading sensors.
// The effect of temperature has not been taken into account so I can't
promise that it will work if you
// calibrate indoors and then use it outdoors. Best is to calibrate and use at
the same room temperature.

/* ==========  LICENSE  ================================
 I2Cdev device library code is placed under the MIT license
 Copyright (c) 2011 Jeff Rowberg

 Permission is hereby granted, free of charge, to any person obtaining a copy
 of this software and associated documentation files (the "Software"), to deal
 in the Software without restriction, including without limitation the rights
 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 copies of the Software, and to permit persons to whom the Software is
 furnished to do so, subject to the following conditions:

 The above copyright notice and this permission notice shall be included in
 all copies or substantial portions of the Software.

 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 THE SOFTWARE.
 ===========================================================
```

```
 */

// I2Cdev and MPU6050 must be installed as libraries
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

///////////////////////////////   CONFIGURATION   //////////////////////
/////
//Change these 3 variables if you want to fine tune the sketch to your needs.
int buffersize=1000;     //Amount of readings used to average, make it higher
to get more precision but sketch will be slower  (default:1000)
int acel_deadzone=8;     //Acelerometer error allowed, make it lower to get
more precision, but sketch may not converge  (default:8)
int giro_deadzone=1;     //Giro error allowed, make it lower to get more
precision, but sketch may not converge  (default:1)

// default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for InvenSense evaluation board)
// AD0 high = 0x69
//MPU6050 accelgyro;
MPU6050 accelgyro(0x68); // <-- use for AD0 high

int16_t ax, ay, az,gx, gy, gz;

int mean_ax,mean_ay,mean_az,mean_gx,mean_gy,mean_gz,state=0;
int ax_offset,ay_offset,az_offset,gx_offset,gy_offset,gz_offset;

///////////////////////////////   SETUP   ////////////////////////////////
////
void setup() {
  // join I2C bus (I2Cdev library doesn't do this automatically)
  Wire.begin();
  // COMMENT NEXT LINE IF YOU ARE USING ARDUINO DUE
  TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz). Leonardo measured
250kHz.

  // initialize serial communication
  Serial.begin(115200);

  // initialize device
  accelgyro.initialize();

  // wait for ready
  while (Serial.available() && Serial.read()); // empty buffer
  while (!Serial.available()){
    Serial.println(F("Send any character to start sketch.\n"));
```

```
    delay(1500);
  }
  while (Serial.available() && Serial.read()); // empty buffer again

  // start message
  Serial.println("\nMPU6050 Calibration Sketch");
  delay(2000);
  Serial.println("\nYour MPU6050 should be placed in horizontal position, with
package letters facing up. \nDon't touch it until you see a finish
message.\n");
  delay(3000);
  // verify connection
  Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful"
: "MPU6050 connection failed");
  delay(1000);
  // reset offsets
  accelgyro.setXAccelOffset(0);
  accelgyro.setYAccelOffset(0);
  accelgyro.setZAccelOffset(0);
  accelgyro.setXGyroOffset(0);
  accelgyro.setYGyroOffset(0);
  accelgyro.setZGyroOffset(0);
}

////////////////////////////////   LOOP   ////////////////////////////////
///
void loop() {
  if (state==0){
    Serial.println("\nReading sensors for first time...");
    meansensors();
    state++;
    delay(1000);
  }

  if (state==1) {
    Serial.println("\nCalculating offsets...");
    calibration();
    state++;
    delay(1000);
  }

  if (state==2) {
    meansensors();
    Serial.println("\nFINISHED!");
    Serial.print("\nSensor readings with offsets:\t");
    Serial.print(mean_ax);
    Serial.print("\t");
    Serial.print(mean_ay);
```

```
    Serial.print("\t");
    Serial.print(mean_az);
    Serial.print("\t");
    Serial.print(mean_gx);
    Serial.print("\t");
    Serial.print(mean_gy);
    Serial.print("\t");
    Serial.println(mean_gz);
    Serial.print("Your offsets:\t");
    Serial.print(ax_offset);
    Serial.print("\t");
    Serial.print(ay_offset);
    Serial.print("\t");
    Serial.print(az_offset);
    Serial.print("\t");
    Serial.print(gx_offset);
    Serial.print("\t");
    Serial.print(gy_offset);
    Serial.print("\t");
    Serial.println(gz_offset);
    Serial.println("\nData is printed as: acelX acelY acelZ giroX giroY
giroZ");
    Serial.println("Check that your sensor readings are close to 0 0 16384 0 0
0");
    Serial.println("If calibration was succesful write down your offsets so
you can set them in your projects using something similar to
mpu.setXAccelOffset(youroffset)");
    while (1);
  }
}

/////////////////////////////////   FUNCTIONS   ////////////////////////////
////////
void meansensors(){
  long i=0,buff_ax=0,buff_ay=0,buff_az=0,buff_gx=0,buff_gy=0,buff_gz=0;

  while (i<(buffersize+101)){
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    if (i>100 && i<=(buffersize+100)){ //First 100 measures are discarded
      buff_ax=buff_ax+ax;
      buff_ay=buff_ay+ay;
      buff_az=buff_az+az;
      buff_gx=buff_gx+gx;
      buff_gy=buff_gy+gy;
      buff_gz=buff_gz+gz;
    }
```

```
    if (i==(buffersize+100)){
      mean_ax=buff_ax/buffersize;
      mean_ay=buff_ay/buffersize;
      mean_az=buff_az/buffersize;
      mean_gx=buff_gx/buffersize;
      mean_gy=buff_gy/buffersize;
      mean_gz=buff_gz/buffersize;
    }
    i++;
    delay(2); //Needed so we don't get repeated measures
  }
}

void calibration(){
  ax_offset=-mean_ax/8;
  ay_offset=-mean_ay/8;
  az_offset=(16384-mean_az)/8;

  gx_offset=-mean_gx/4;
  gy_offset=-mean_gy/4;
  gz_offset=-mean_gz/4;
  while (1){
    int ready=0;
    accelgyro.setXAccelOffset(ax_offset);
    accelgyro.setYAccelOffset(ay_offset);
    accelgyro.setZAccelOffset(az_offset);

    accelgyro.setXGyroOffset(gx_offset);
    accelgyro.setYGyroOffset(gy_offset);
    accelgyro.setZGyroOffset(gz_offset);

    meansensors();
    Serial.println("...");

    if (abs(mean_ax)<=acel_deadzone) ready++;
    else ax_offset=ax_offset-mean_ax/acel_deadzone;

    if (abs(mean_ay)<=acel_deadzone) ready++;
    else ay_offset=ay_offset-mean_ay/acel_deadzone;

    if (abs(16384-mean_az)<=acel_deadzone) ready++;
    else az_offset=az_offset+(16384-mean_az)/acel_deadzone;

    if (abs(mean_gx)<=giro_deadzone) ready++;
    else gx_offset=gx_offset-mean_gx/(giro_deadzone+1);

    if (abs(mean_gy)<=giro_deadzone) ready++;
    else gy_offset=gy_offset-mean_gy/(giro_deadzone+1);
```

```
    if (abs(mean_gz)<=giro_deadzone) ready++;
    else gz_offset=gz_offset-mean_gz/(giro_deadzone+1);

    if (ready==6) break;
  }
}
        //**********************CODE END**********************
```

The values of the calibration readings were:

- giroX = 4
- giroY = −6
- giroZ = 17
- acelZ = 1958

4.2 Self-Balance Code

Code:

```
#include <PID_v1.h>
#include <LMotorController.h>
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

#define MIN_ABS_SPEED 20

MPU6050 mpu;

bool dmpReady = false;
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];

Quaternion q;
VectorFloat gravity;
float ypr[3];


double originalSetpoint = 185.5;
double setpoint = originalSetpoint;
double movingAngleOffset = 0.1;
```

```
double input, output;
int moveState = 0;
double Kp = 70;
double Kd = 2;
double Ki = 150;
PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

double motorSpeedFactorLeft = 0.6;
double motorSpeedFactorRight = 0.62;

int ENA = 3;
int IN1 = 10;
int IN2 = 9;
int IN3 = 6;
int IN4 = 5;
int ENB = 11;
LMotorController motorController(ENA, IN1, IN2, ENB, IN3, IN4,
motorSpeedFactorLeft, motorSpeedFactorRight);


long time1Hz = 0;
long time5Hz = 0;

volatile bool mpuInterrupt = false;
void dmpDataReady()
{
  mpuInterrupt = true;
}


void setup()
{

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
  Wire.begin();
  TWBR = 24;
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
  Fastwire::setup(400, true);
#endif


  Serial.begin(115200);
  while (!Serial);


  Serial.println(F("Initializing I2C devices..."));
  mpu.initialize();
```

```
  Serial.println(F("Testing device connections..."));
  Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") :
F("MPU6050 connection failed"));


  Serial.println(F("Initializing DMP..."));
  devStatus = mpu.dmpInitialize();

// Values of MPU6050 Calibration are entered in the following four lines:
  mpu.setXGyroOffset(4);
  mpu.setYGyroOffset(-6);
  mpu.setZGyroOffset(17);
  mpu.setZAccelOffset(1958);

  if (devStatus == 0)
  {

    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    Serial.println(F("Enabling interrupt detection (Arduino external interrupt
0)..."));
    attachInterrupt(0, dmpDataReady, RISING);
    mpuIntStatus = mpu.getIntStatus();


    Serial.println(F("DMP ready! Waiting for first interrupt..."));
    dmpReady = true;


    packetSize = mpu.dmpGetFIFOPacketSize();


    pid.SetMode(AUTOMATIC);
    pid.SetSampleTime(10);
    pid.SetOutputLimits(-255, 255);
  }
  else
  {

    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
  }
}


void loop()
{
```

```
  if (!dmpReady) return;


  while (!mpuInterrupt && fifoCount < packetSize)
  {


    pid.Compute();
    motorController.move(output, MIN_ABS_SPEED);

  }


  mpuInterrupt = false;
  mpuIntStatus = mpu.getIntStatus();


  fifoCount = mpu.getFIFOCount();


  if ((mpuIntStatus & 0x10) || fifoCount == 1024)
  {

    mpu.resetFIFO();
    Serial.println(F("FIFO overflow!"));


  }
  else if (mpuIntStatus & 0x02)
  {

    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();


    mpu.getFIFOBytes(fifoBuffer, packetSize);

    fifoCount -= packetSize;

    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
#if LOG_INPUT
    Serial.print("ypr\t");
    Serial.print(ypr[0] * 180 / M_PI);
    Serial.print("\t");
    Serial.print(ypr[1] * 180 / M_PI);
    Serial.print("\t");
    Serial.println(ypr[2] * 180 / M_PI);
#endif
```

```
    input = ypr[1] * 180 / M_PI + 180;
  }
}
        //**********************CODE END**********************
```

The code starts by defining some constants and variables that will be used later in the program. For example, the MIN_ABS_SPEED constant is set to 20, which is the minimum absolute speed that the motors can be set to, and the originalSetpoint variable is set to 185.5°, which is the desired angle that the robot should maintain.

Next, the code initializes the MPU6050 sensor by calling the mpu.initialize() function, which sets up the I2C communication with the sensor. It then checks if the sensor is connected properly by calling the mpu.testConnection() function and printing the result to the serial monitor.

The code then initializes the DMP (Digital Motion Processor) of the MPU6050 sensor by calling the mpu.dmpInitialize() function. The DMP allows the sensor to perform complex calculations such as fusion of accelerometer and gyroscope data to provide more accurate sensor readings. The code also sets some sensor calibration values by calling functions such as mpu.setXGyroOffset() and mpu.setZAccelOffset().

If the DMP initialization is successful, the code enables the DMP and interrupt detection by calling mpu.setDMPEnabled() and attachInterrupt() functions, respectively. The PID control loop is also initialized by creating a PID object and setting its parameters such as Kp, Ki, and Kd.

In the main loop, the code checks if the DMP is ready and waits for an interruption. Once an interruption occurs, the code reads sensor data from the MPU6050 sensor and computes the PID output using the pid.Compute() function. The output is then passed to the motorController.move() function, which controls the speed of the motors.

In this code we changed the setpoint value of the MPU6050 from 180° in the original code to 185.5° by trial-and-error method. Also, the right motor speed factor was changed from 0.5 to 0.62 by observing the speed of the right motor relative to the left motor until both motors were almost the same speed.

Finally, PID values were changed (tuned) using trial and error method as we will discuss in the next section.

5 PID Tuning

Proportional Integral Derivative (PID) controller is an automatically optimized and accurate control system used to regulate different parameters like temperature, pressure, and speed at desired values. The basic mechanism used in PID controllers is control loop feedback. A PID calculates the error by calculating difference between actual value and desired value and then sets the deciding parameters accordingly. This error is continuously being calculated until the process stops.

It consists of three parameters (P), (I), (D). The combination of proportional, integral and derivative controls in the PID control system has a specific purpose. Proportional control excels at fast rise time, while integral control can eliminate the steady-state error. Moreover, derivative control can reduce overshoot. When we combined them, we get the result of control with error-eliminating properties, reducing in rise time, settling time, and overshoot.

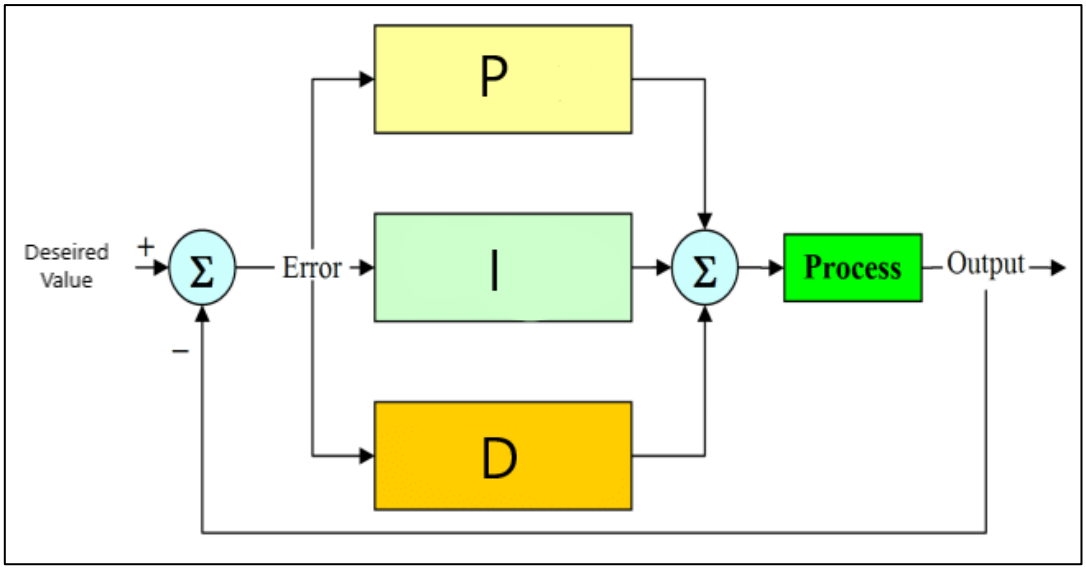

Fig. 5.1: PID Control System

5.1 PID Parameters

5.1.1 Proportional (P) parameter

The P-action is proportional to the error or the PV. The error (or PV) is multiplied with the proportional gain and added to the controller output. The P-action gives the output a 'kick' in the right direction. If the error value is zero, then the P action is zero. This implies that a controller with only P action needs a non-zero error to have a non-zero output. Accurate tracking is therefore not possible with only P control.

5.1.2 Integral (I) parameter

Typically, the I-action will act much slower compared to the proportional action. However, it will bring the error to zero eventually, which proportional action can't do. So basically, the integral action looks at the past and checks if the error is getting to the setpoint. If not, it's acting on the output. It is going to steer the wheel until you are heading in the intended direction.

5.1.3 Derivative (D) parameter

The integral doesn't have the possibility to predict the behavior of error. The derivative action addresses this problem by anticipating the future behavior of the error. So, the derivative action is the change of the error. It adds a contribution to the output according to how the error changes. When the error is positive, but is starting to decline the D action, it will reduce the output of the controller. It's the brake that tries to avoid overshooting. It reduces the oscillations induced by the other two actions. It can speed up the controller to the setpoint that we want to achieve. It reduces the oscillations induced by the other two actions. It can speed up the controller to the setpoint that you want to reach. However, the derivative action is not often used in PID tuning. The problem is that it can amplify noise. If the error signal is very noisy, the controller output tends to oscillate a lot.

Table 5.1: Effects of control parameters on the close loop system

Closed loop responce	Rise time	Over shoot	Settling time	Steady state error
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	eliminate
Kd	Small Change	Decrease	Decrease	No change

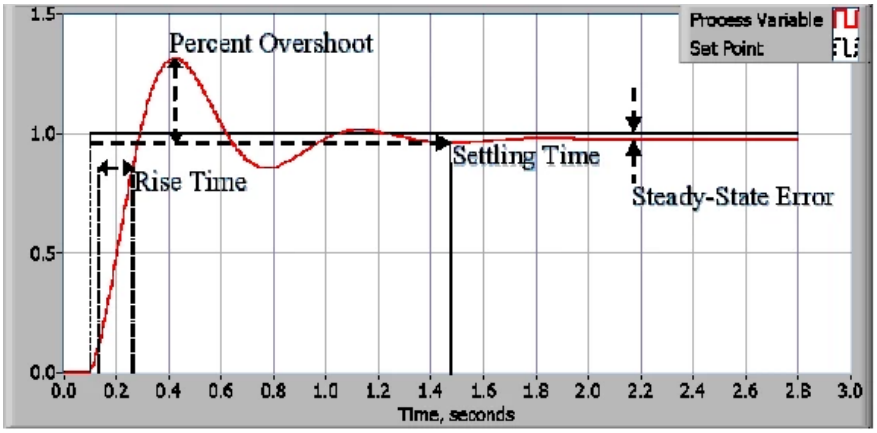

Fig. 5.2: Typical Response curve for a step input

5.2 The process of PID tuning

1- Set K_i and K_d to zero and gradually increase K_p so that the robot starts to oscillate about the zero position.

2- Increase K_i so that the response of the robot is faster when it is out of balance. K_i should be large enough so that the angle of inclination does not increase. The robot should come back to zero position if it is inclined.

3- Increase K_d to reduce the oscillations. The overshoots should also be reduced by now.

4- Repeat the above steps by fine tuning each parameter to achieve the desired result.

5.3 PID Tuning Iterations

Iteration	K_p	K_i	K_d	Observation
1	150	0	0	Jittering starts to occur
2	25	0	0	Robot doesn't balance itself
3	90	0	0	Robot almost balance itself with a lot of human help
4	100	0	0	Robot almost balance itself with little human help
5	100	300	0	Robot settling time was large
6	100	50	0	Robot almost balance but then go in one direction
7	100	40	0	Slow response
8	100	50	0.1	Robot starts to move randomly
9	100	50	10	Robot jitters again
10	100	50	6	Jittering is still occurring
11	100	50	2	Robot fails to balance
12	80	75	2	Robot starts to balance but still not good
13	70	100	2	Robot balances but get further from its initial positions
14	70	150	2	Robot balance is very good (Accepted)

6 Transfer Function

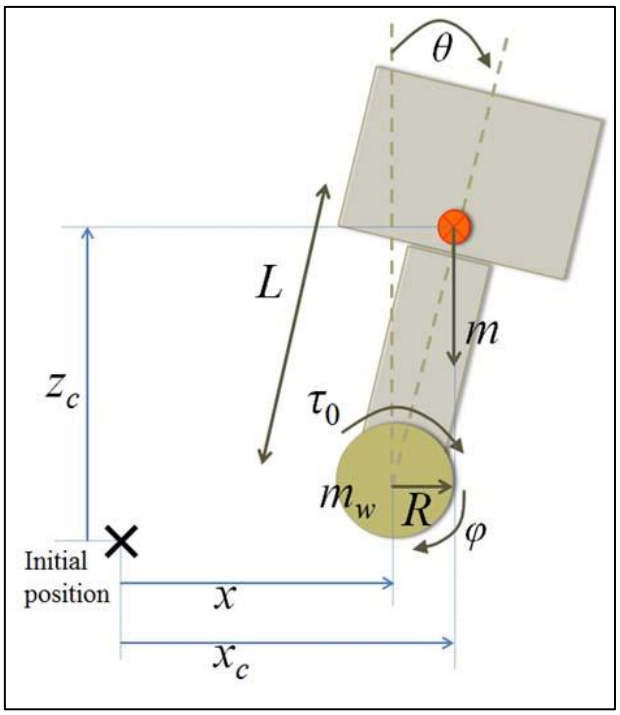

Fig. 6.1: Schematic of balancing robot with relevant parameters

6.1 Derivation of the Dynamic Equations of Motion

We used the Lagrange's method to derive the dynamic equations for this system. We can write $x, x_c . z_c . \dot{x},\ \dot{x}_c, \dot{z}_c$ as follows:

$$x = R\varphi$$
$$x_c = R\varphi + L\sin\theta$$
$$z_c = R + L\cos\theta$$

$$\dot{x} = R\dot{\varphi}$$
$$\dot{x}_c = R\dot{\varphi} + L\dot{\theta}\cos\theta$$
$$\dot{z}_c = -L\dot{\theta}\sin\theta$$

Continuing, we can write the potential energy E_p (zero is defined as the potential energy at the upright position) and the kinetic energy E_k as follows:

$$E_p = mg(R + L\cos\theta) - mg(R + L) = mgL(\cos\theta - 1)$$

$$E_k = \frac{1}{2}m_w\dot{x}^2 + \frac{1}{2}I_w\dot{\varphi}^2 + \frac{1}{2}m\dot{x}_c^{\,2} + \frac{1}{2}m\dot{z}_c^{\,2} + \frac{1}{2}I\dot{\theta}^2$$

$$= \frac{1}{2}\left(I_w + m_w R^2 + mR^2\right)\dot{\varphi}^2 + mRL\cos\theta\dot{\varphi}\dot{\theta} + \frac{1}{2}\left(I + mL^2\right)\dot{\theta}^2$$

6.2 Derivation of the Lagrangian

Lagrangian ℓ can be written as the difference between the kinetic energy $E_k$and the potential energy E_p.

$$\mathcal{L} = E_k - E_p$$

$$= \frac{1}{2}\left(I_w + m_w R^2 + mR^2\right)\dot{\varphi}^2 + mRL\cos\theta\dot{\varphi}\dot{\theta} + \frac{1}{2}\left(I + mL^2\right)\dot{\theta}^2 - mgL(\cos\theta - 1)$$

Therefore, the dynamic equations for φ-coordinate and θ-coordinate can be derived as follows:

(i) φ-coordinate

$$\frac{\partial \mathcal{L}}{\partial \dot{\varphi}} = \left(I_w + m_w R^2 + mR^2\right)\dot{\varphi} + mRL\,\dot{\theta}\cos\theta$$

$$\frac{\partial \mathcal{L}}{\partial \varphi} = 0$$

$$\therefore \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\varphi}}\right) - \frac{\partial \mathcal{L}}{\partial \varphi} = \left(I_w + m_w R^2 + mR^2\right)\ddot{\varphi} + mRL\cos\theta\ddot{\theta} - mRL\sin\theta\dot{\theta}^2 = \mu$$

(ii) θ-coordinate

$$\frac{\partial \mathcal{L}}{\partial \dot{\theta}} = mRL\cos\theta\dot{\varphi} + \left(I + mL^2\right)\dot{\theta}$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = -mRL\sin\theta\dot{\varphi}\dot{\theta} + mgL\sin\theta$$

$$\therefore \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) - \frac{\partial \mathcal{L}}{\partial \theta} = \left(I + mL^2\right)\ddot{\theta} + mRL\cos\theta\ddot{\varphi} - mRL\sin\theta = \chi$$

Where μ and χ are generalized forces (torques) for each coordinate.

7 Recommendations for Improving the System

In this project there were better components and ideas that we might have used but couldn’t use because of its high price and complexity.

7.1 Ideas

Here are some ideas that we might have added to our project.

7.1.1 Self-Balance Robot with Line following

In automated warehouses a two-wheeled line following robot might be used to stack items in shelves. This is done by integrating a self-balancing robot with IR line tracking sensor (TCRT5000), the robot balances itself while following a black line on the ground to deliver items to shelves.

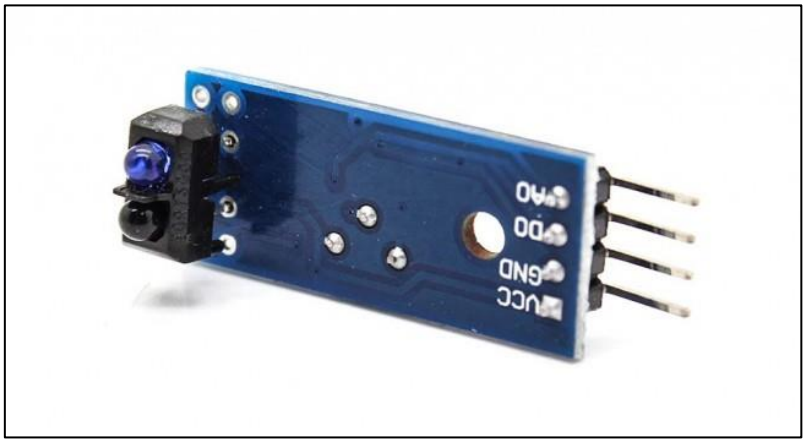

Fig. 7.1: TCRT5000

This idea can also be used in automated restaurants where two wheeled servant robots deliver food to tables, but in this case the robots must be equipped by an ultrasonic sensor to detect if there is someone or something in front of the robot, so it doesn’t hit the obstacle ahead.

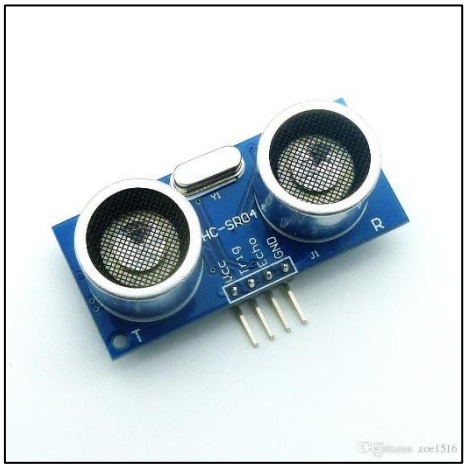

Fig. 7.2: Ultrasonic Sensor

7.1.2 Self-Balance Robot with Bluetooth

To control any wheeled robot using a remote control or a mobile device, a Bluetooth module must be used. For the self-balancing robot, it's the same case.

A Bluetooth module is used to receive signals from a mobile phone and then send signals to the microcontroller to send an actuating signal to the motors, the robot balances itself while also moving.

7.2 Components

7.2.1 Sensor

Our MPU6050 sensor is an excellent choice for our project due to its price and its good accuracy, but there are better sensors than ours but for more complicated projects.

- **MPU9250:** this sensor module combines a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer in a single package. (LE 360.00)

- **BNO055:** this sensor module offers a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer, along with a built-in processor for fusion of sensor data. ($33 + shipping)

A magnetometer is a device that measures magnetic field or magnetic dipole moment. If our project was self-balancing and self-parking car, we would have needed MPU9250 or BNO055 because its magnetometer will detect metal objects (other parked cars) and will avoid hitting those cars.

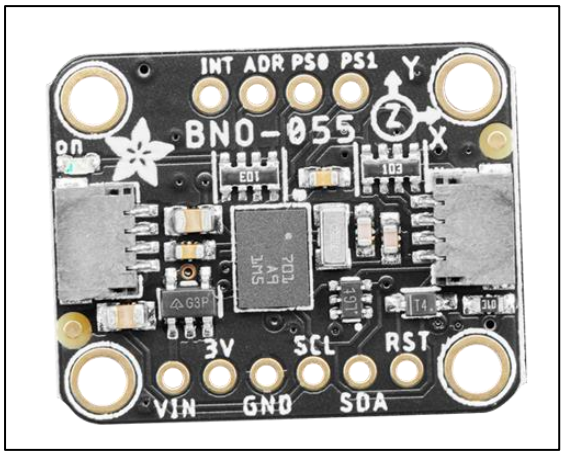

Fig. 7.4: BNO055

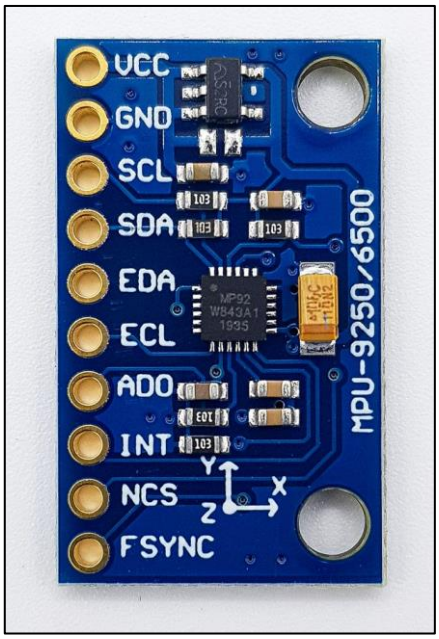

Fig. 7.3: MPU9250

7.2.2 Actuator

A Stepper motor instead of our Yellow DC Geared motor was the best choice of motor that we could have used for a self-balancing robot due to its high holding torque, but the price of this motor equals 380 L.E and we need two. Its driver module would be A4988 Stepper driver IC instead of L298N driver module.

Fig. 7.5: Stepper Motor

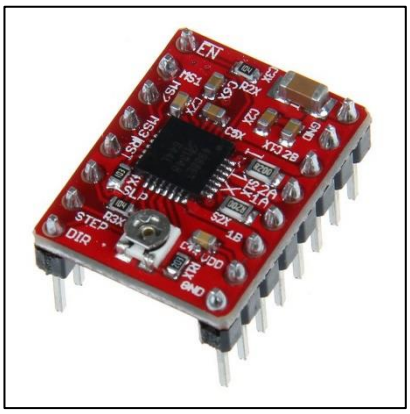

Fig. 7.6: A4988 Driver Module

7.2.3 Wheels

Wheels must have a good grip over the floor so that the robot doesn't slip. The wheel we used is not the best option, but it gets the job done. A better option though would be wheels with teeth like the wheels in Fig. 6.9 and Fig. 6.10.

Fig. 7.8: 130 mm Metal Coupler Wheel [210E GP/piece]

Fig. 7.7: 100 mm High Load Robotic Wheel [105 EGP/piece]

7.2.4 Microcontroller

The Arduino UNO microcontroller is a very good choice for our project. However, a better choice would be Arduino NANO due to its smaller size and its less weight compared to Arduino UNO.

We didn’t use Arduino NANO despite being cheaper because it is more difficult to handle and use as it requires outer pins to be wielded to it, also it can get damaged easily.

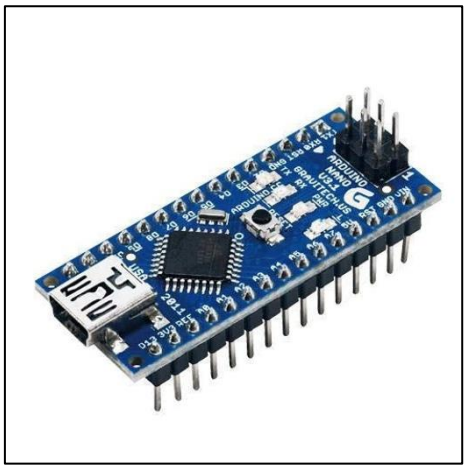

Fig. 7.9: Arduino NANO

References

[1] Curiel-Olivares, G., Linares-Flores, J., Guerrero-Castellanos, J. F., & Hernández-Méndez, A. (2021). Self-balancing based on active disturbance rejection controller for the two-in-wheeled electric vehicle, experimental results. Mechatronics, 76, 102552. https://doi.org/10.1016/j.mechatronics.2021.102552

[2] DC geared motor dual shaft bo motor-straight. Micro Ohm Electronics. (2023, March 10). https://microohm-eg.com/product/dc-geared-motor-dual-shaft-bo-motor-straight/

[3] Ding, Y., Gafford, J., & Kunio, M. (n.d.). Modeling, simulation and fabrication of a balancing robot. https://scholar.harvard.edu/sites/scholar.harvard.edu/files/jgafford/files/finalpaper_final_version.pdf?m=1435603839

[4] DIY self balancing robot using Arduino. Circuit Digest - Electronics Engineering News, Latest Products, Articles and Projects. (n.d.). https://circuitdigest.com/microcontroller-projects/arduino-based-self-balancing-robot

[5] Gonzalez, C., Alvarado, I., & Peña, D. M. (2017). Low cost Two-wheels self-balancing robot for control education. IFAC-PapersOnLine, 50(1), 9174–9179. https://doi.org/10.1016/j.ifacol.2017.08.1729

[6] Hana, Dejan, Sergio, Dheenadayalan, Peyerl, F., Gennetten, E., Caden, Gab, Xuan, Mr., Jaleel, Hashtag, Glover, W., Cesar, Eric, Meyer, E. H., Chatch, Matsobane, Kelly, Houssam, … Victor. (2022, February 17). L298N motor driver - arduino interface, how it works, codes, schematics. How To Mechatronics. https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

[7] How does the MPU6050 Accelerometer & Gyroscope Sensor Work and interfacing it with Arduino. Arduino MPU6050 Tutorial - How MPU6050 Module Works and Interfacing it with Arduino. (n.d.). https://circuitdigest.com/microcontroller-projects/interfacing-mpu6050-module-with-

arduino#:~:text=MPU6050%20is%20an%20Inertial%20Measurement,%2C%20and%20a16%2Dbit%20ARC.

[8] Last Minute Engineers. (2022a, October 16). In-depth: Interface L298n DC Motor driver module with Arduino. Last Minute Engineers. https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/

[9] Last Minute Engineers. (2022b, October 29). In-depth: Interface MPU6050 Accelerometer & Gyroscope sensor with Arduino. Last Minute Engineers. https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/

[10] Self balancing robot. Flyrobo. (n.d.). https://www.flyrobo.in/blog/self-balancing-robot#:~:text=The%20working%20concept%20of%20a%20Self-balancing%20Robot&text=The%20gyroscope%20used%20in%20this,robot%20to%20balance%20itself%20upright.