SELF BALANCING ROBOT

Documentary

SELF BALANCING ROBOT

Team Members:-

Mentors:

Harini

Mayur

Namit

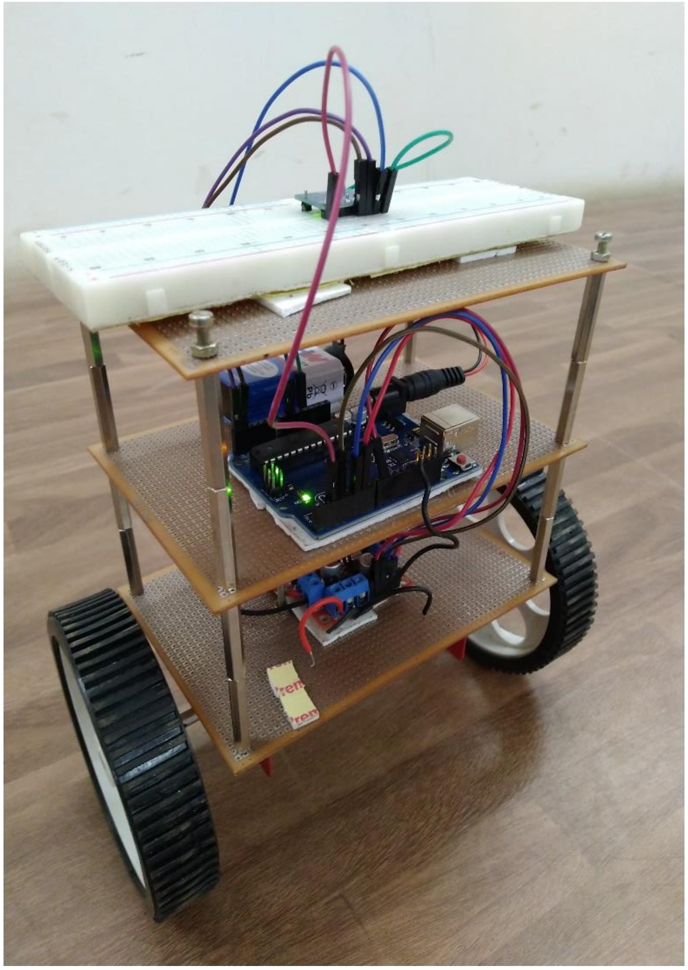
Mentees:

Bhaskar Sahu

Baliram

Vivek

Basavraj

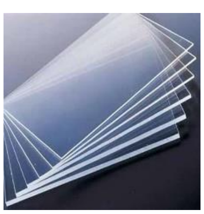


Introduction and Motivation:-

The two wheels self-balancing robot is an important kind of mobile robot. Self-balancing robot has the capability to balance on its two wheels without falling. It works on the principle of inverted pendulum. It can be seen as a inverted pendulum mounted on two wheels. This kind of self-balancing robots can be used in restaurant as a waiter and can be use in offices to take the light material (files and papers) from one place to another. This motivated us to work on this project. Segway self-balancing robot is one of the self-balancing robot.

Material Required:-

Mechanical Parts:-



3x Acrylic sheets

16x Brass stud M4x16

2x DC motor

2x wheels

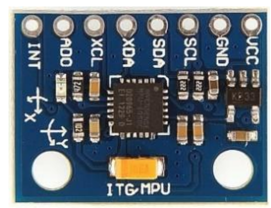
2x Motor bracket

Screw and Nuts

Electronic Modules:-

Arduino UNO

Motor Driver L298N



MPU6050 Gyroscope

Batteries of sufficient power

Jumper wires

Functions of different materials:-

All the mechanical components are used to make the assortment of the robot.

Arduino:-

It receives tilt angle and angular velocity of falling of the robot in analog form from gyroscope (GY-521) and accordingly it provides output to motor driver in digital form.

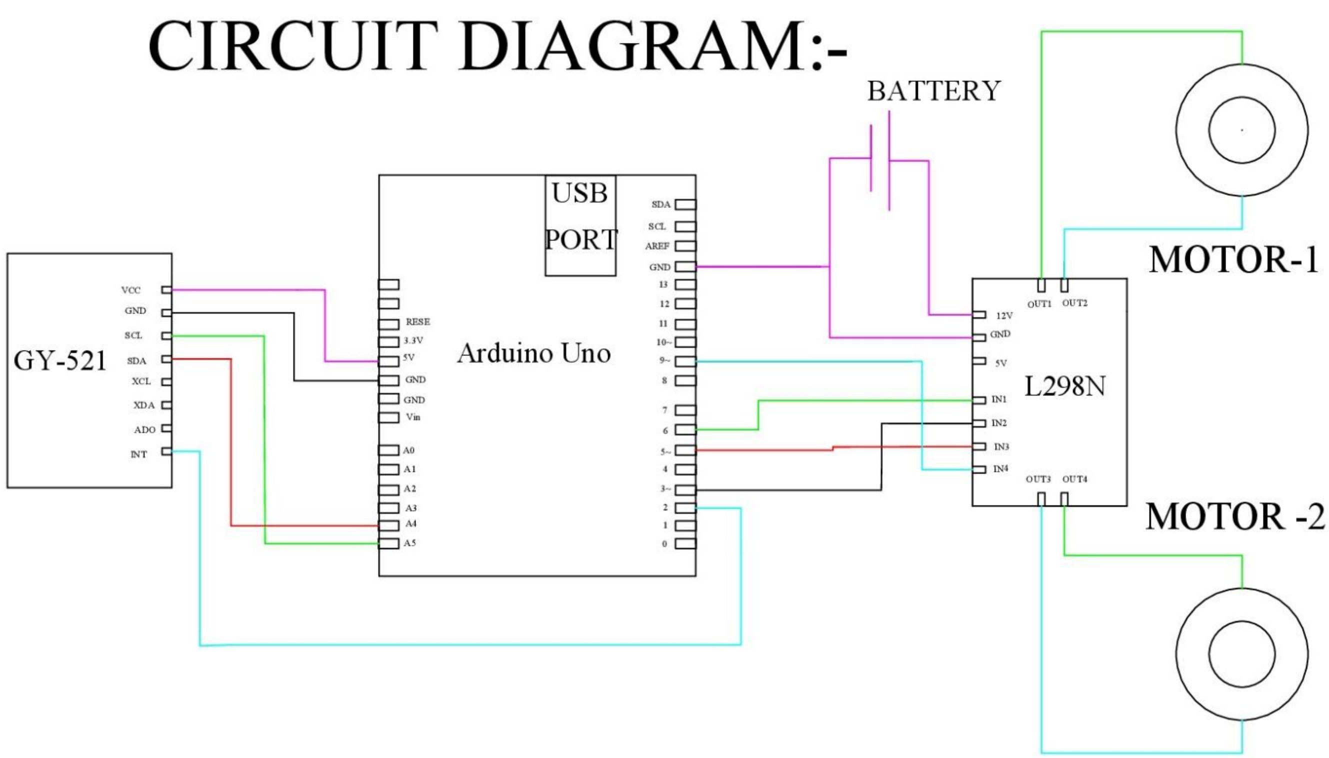
Motor Driver:-

As name suggest it driver the motors, it receives digital signal from the arduino and gives output to the motors.

MPU6050:-

It is the main part of robot. It measure the value of acceleration from which we calculate the tilt angle of robot to the arduino in analog form.

Circuit Diagram:-



Working of Robot:-

Self-balancing robot works on the principle of inverted pendulum. The physics for this robot are simple, the robot stand in two points lined, and it tends to fall and lose his verticality, movement of the wheeel in the direction of the falling rises the robot to recover the verticle position. Gyroscope measures the acceleration of the robot in x, y directions and also the angular velocity of falling of robot. We can find the tilt angle of robot at any time by acceleration. Accordingly the arduino gives the digital output from PWM pins to the motor driver(PWM pins are used to varry the output voltage). The code for the arduino is given at the last of this document.

Process of building Robot:-

1. *Prepare the model of the robot using mechanical components.*
2. *Make all the connections using electrical modules as given it the circuit diagram.*
3. *Keep the robot in stable condition and find the offset values for the robot by uploadig the given codes in the arduino and use these values in the final code(finding offset by computer may take 8-10 mints).*
4. *Tuning of the Robot :- Now adjust the values of Kp, Ki and Kd to balancing the robot.*
5. *Kp is directly proportional to the ossillation of robot , Response time of the motors incraese by decreasing the Kd(its value is generally less than 3), Ki has little effect unless it is extreme.*

Points to be noted:-

1. *While calculating the offset values fix position of robot vertically stable and don't disturb the position of the robot till the offset has calculated.*
2. *Fix the components (aduino board, battery,etc) in such a way that vertical line through the center of gravity of model intersect the line joining the two motors.*
3. *Don't give more that 3.3V voltage to the gyroscope.*

Code to find the offset:-

```
// I2Cdev and MPU6050 must be installed as libraries

#include "I2Cdev.h"

#include "MPU6050.h"

#include "Wire.h"


///////////////////////////////////   CONFIGURATION   /////////////////////////////

//Change this 3 variables if you want to fine tune the skecth to your needs.

int buffersize=1;     //Amount of readings used to average, make it higher to get more precision but sketch will be slower  (default:1000)

int acel_deadzone=20;     //Acelerometer error allowed, make it lower to get more precision, but sketch may not converge  (default:8)

int giro_deadzone=10;     //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(200);
  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(300);
  // verify connection
  Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful" : "MPU6050 connection 
failed");
  delay(10);
  // 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(10);
  }

  if (state==1) {
```

```
  Serial.println("\nCalculating offsets...");
  calibration();
  state++;
  delay(10);
 }

 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+51)){
    // read raw accel/gyro measurements from device
    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    if (i>50 && i<=(buffersize+50)){ //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+50)){
```

```
      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;
    }
  }
```

Complete code for self balancing:-

```
#include "Wire.h"

#include "I2Cdev.h"

#include "MPU6050.h"

#include "math.h"


#define leftMotorPWMPin   3

#define leftMotorDirPin   6

#define rightMotorPWMPin  5

#define rightMotorDirPin  9


#define Kp  30

#define Kd  0.1

#define Ki  60

#define sampleTime  0.005

#define targetAngle -2.5


MPU6050 mpu;


int16_t accX, accZ, gyroY;

volatile int motorPower, gyroRate, mp2;

volatile float accAngle, gyroAngle, currentAngle, prevAngle=0, error, prevError=0, errorSum=0;


void setMotors(int leftMotorSpeed, int rightMotorSpeed) {

  if(leftMotorSpeed >= 0) {

    analogWrite(leftMotorPWMPin, 0.7*leftMotorSpeed);

    digitalWrite(leftMotorDirPin, LOW);
```

```
  }
  else {
    analogWrite(leftMotorPWMPin, 255 + leftMotorSpeed);
    digitalWrite(leftMotorDirPin, HIGH);
  }
  if(rightMotorSpeed >= 0) {
    analogWrite(rightMotorPWMPin, 0.7*rightMotorSpeed);
    digitalWrite(rightMotorDirPin, LOW);
  }
  else {
    analogWrite(rightMotorPWMPin, 255 + rightMotorSpeed);
    digitalWrite(rightMotorDirPin, HIGH);
  }
}

void init_PID() {
  // initialize Timer1
  cli();          // disable global interrupts
  TCCR1A = 0;     // set entire TCCR1A register to 0
  TCCR1B = 0;     // same for TCCR1B
  // set compare match register to set sample time 5ms
  OCR1A = 9999;
  // turn on CTC mode
  TCCR1B |= (1 << WGM12);
  // Set CS11 bit for prescaling by 8
  TCCR1B |= (1 << CS11);
  // enable timer compare interrupt
  TIMSK1 |= (1 << OCIE1A);
```

```
  sei();          // enable global interrupts
}

void setup() {
  Serial.begin(9600);
  // set the motor control and PWM pins to output mode
  pinMode(leftMotorPWMPin, OUTPUT);
  pinMode(leftMotorDirPin, OUTPUT);
  pinMode(rightMotorPWMPin, OUTPUT);
  pinMode(rightMotorDirPin, OUTPUT);
  // set the status LED to output mode

  // initialize the MPU6050 and set offset values, found by applying the above code to your
model
  mpu.initialize();
  mpu.setXAccelOffset(-1050);
  mpu.setZAccelOffset(4820);
  mpu.setYGyroOffset(42);
  // initialize PID sampling loop
  init_PID();

}

void loop() {
  // read acceleration and gyroscope values
  accX = mpu.getAccelerationX();
  accZ = mpu.getAccelerationZ();
  gyroY = mpu.getRotationY();
  // set motor power after constraining it
```

```
  motorPower = constrain(motorPower, -255, 255);
  Serial.println(motorPower);
  setMotors(motorPower, motorPower);}
// The ISR will be called every 5 milliseconds
ISR(TIMER1_COMPA_vect){
  // calculate the angle of inclination
  accAngle = atan2(accX, accZ)*RAD_TO_DEG;
  gyroRate = map(gyroY, -32768, 32767, -250, 250);
  gyroAngle = (float)gyroRate*sampleTime;
  currentAngle = 0.9934*(prevAngle + gyroAngle) + 0.0066*(accAngle);
  error = currentAngle - targetAngle;
  errorSum = errorSum + error;
  errorSum = constrain(errorSum, -300, 300);
  //calculate output from P, I and D values
  motorPower = Kp*(error) + Ki*(errorSum)*sampleTime - Kd*(currentAngle-
prevAngle)/sampleTime;
  prevAngle = currentAngle;

}
```