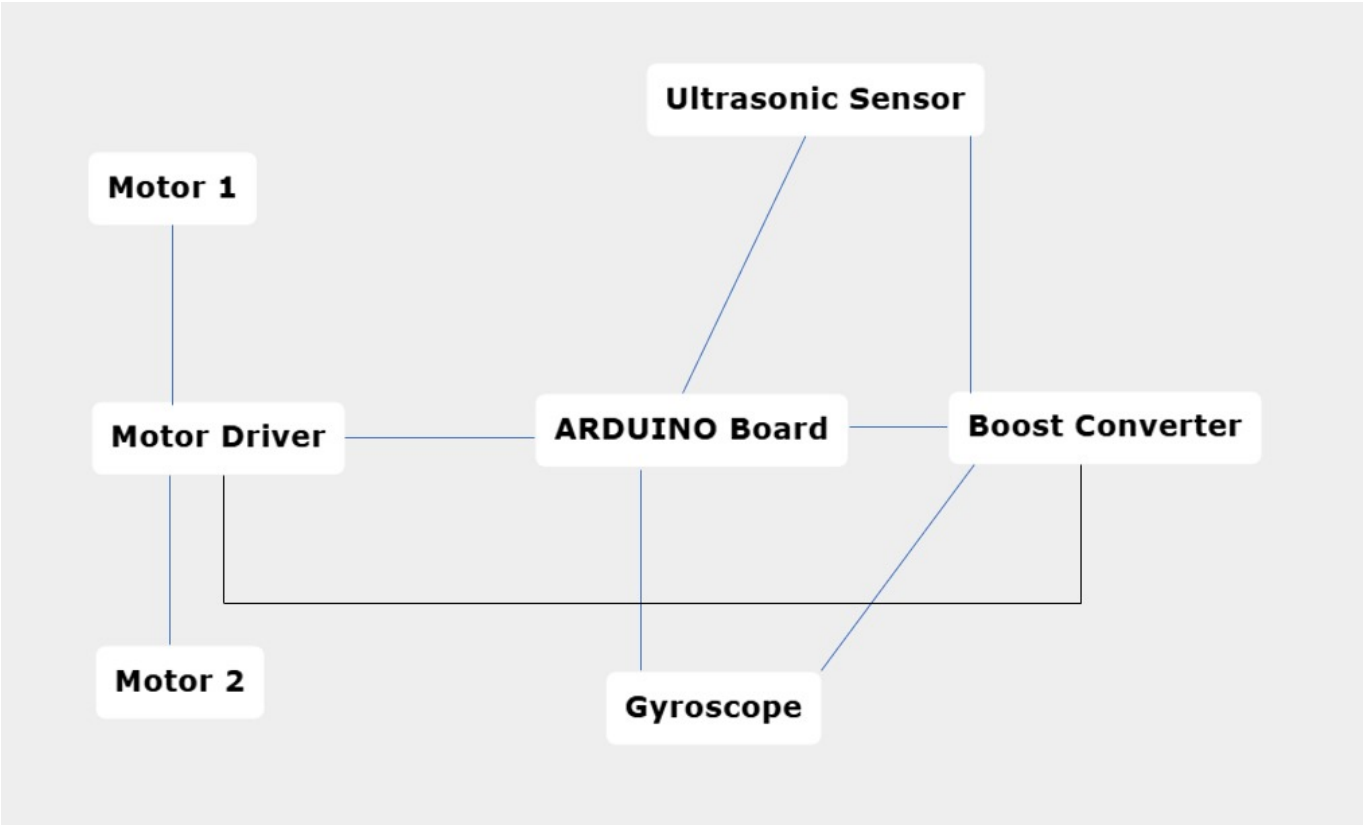
Self-Balancing Robot

Introduction

Self-balancing robot capable of moving while avoiding obstacles. This is a small-sized robot based on the Arduino Pro Mini development board and the MPU6050 accelerometer-gyroscope module.

It will help me understand the concept of self-balancing for a future OFF-ROAD robot project. I will use the PID library for this purpose.

General Description



MPU9250 (Accelerometer + Gyroscope + Magnetometer Module)

Connected to the Arduino via I²C:

- SCL → A5 (SCL)
- SDA → A4 (SDA)
- Provides motion, tilt, and orientation data for balance control.

Arduino Pro Mini

- Central microcontroller managing sensor input and motor control.
- Receives data from MPU9250 and controls the motor driver.
- Pins 2, 3, 4, 5, 6, 7, 8, and 9 are used for communication and control.

TB6612FNG (Motor Driver Module)

- Receives control signals from Arduino to drive the left and right motors.
- Outputs:
- BOUT1, BOUT2 → Left Motor
- AOUT1, AOUT2 → Right Motor

HC-SR04 (Ultrasonic Distance Sensor)

- Connected to Arduino for obstacle detection.
- Uses TRIG and ECHO pins for distance measurements.
- Helps in obstacle avoidance while balancing.

5V Regulators (x2)

- Regulate power from the battery to provide a stable 5V supply.
- Connected to battery terminals via red (+) and black (–) lines.

Battery (NCR18650)

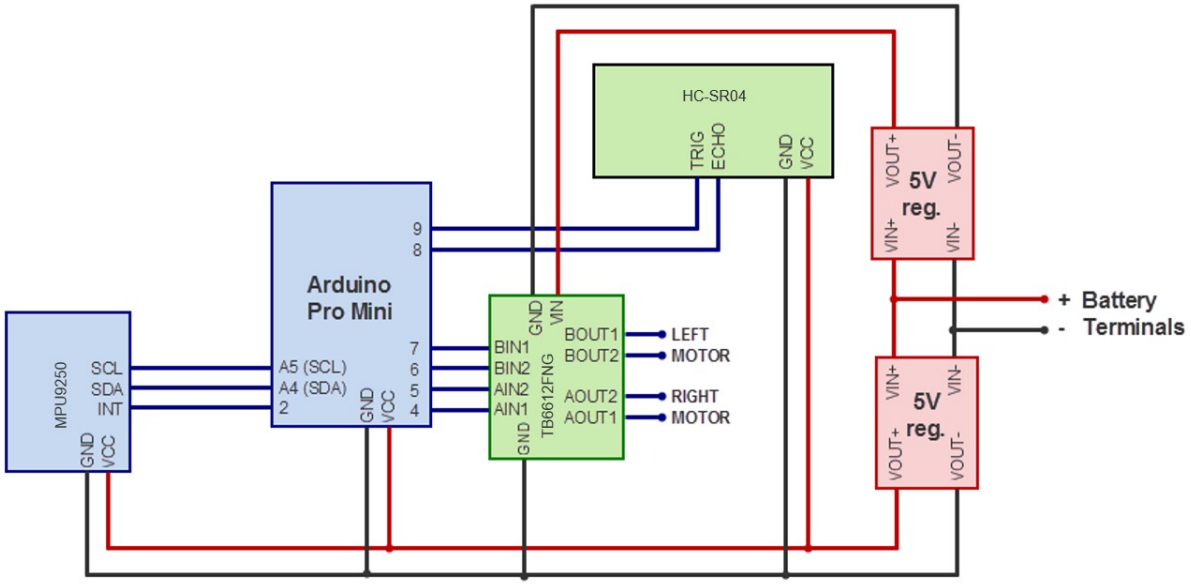
- Provides the primary power source for all components via terminals.
- Positive and negative rails are color-coded: red for +, black for –.

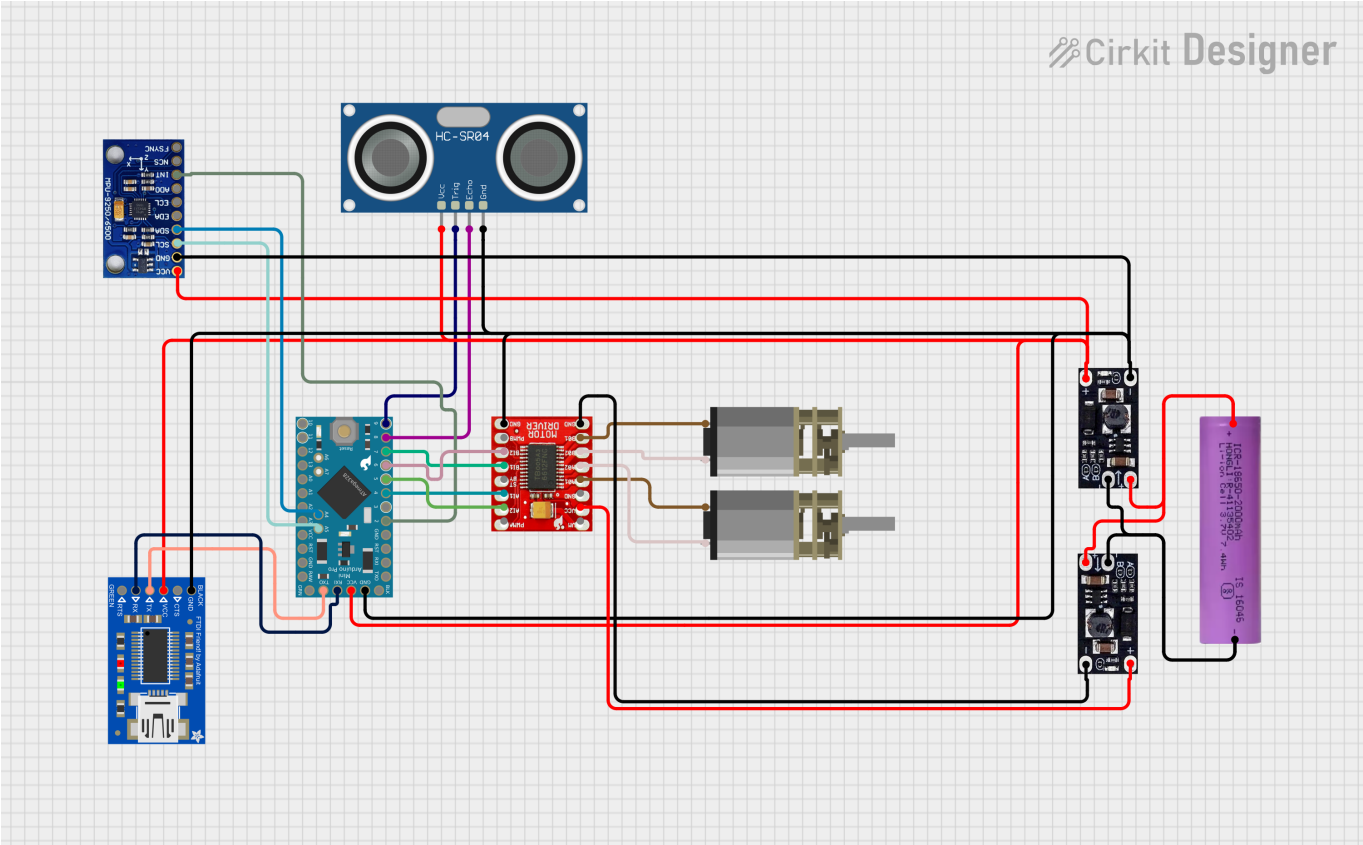
Functional Flow The MPU9250 senses tilt, acceleration, and orientation, sending data to the Arduino, which processes it using a PID algorithm.

Based on the tilt angle, Arduino sends commands to the TB6612FNG motor driver to adjust the rotation of the left and right motors to balance the robot.

The HC-SR04 sensor detects nearby obstacles and is used to alter motor behavior to avoid collisions.

Hardware Design





Name	Description
Arduino Pro Mini	Controller for modules and central processing unit
MPU9250 Module (accelerometer + gyroscope + magnetometer)	Detects tilt, acceleration, and orientation for balance control
TB6612FNG Motor Driver	Drives left and right motors based on Arduino commands
HC-SR04 Distance Sensor	Measures distance to obstacles for collision avoidance
5V Regulators (2x)	Stabilize voltage supply for Arduino, sensors, and motors
NCR18650 Battery	Main power source for the entire system

DC Motors	Provide motion to balance and move the robot
Jumper Wires	Electrical connections between components
XH2.54 Connectors	Secure and detachable module connections

Software Design

Mediu de dezvoltare:

- PlatformIO (bazat pe VS Code), cu suport pentru Arduino UNO.

Biblioteci utilizate:

- I2Cdevlib – pentru comunicarea cu senzorul MPU6050 (accelerometru + giroscop).
- NewPing – pentru măsurarea distanței cu senzorul ultrasonic HC-SR04.

Biblioteci standard:

- Wire.h, math.h.

Algoritmi și funcționalitate:

- Controlul echilibrului se realizează cu un algoritm PID executat într-un ISR hardware (Timer1, 5ms).
- Unghiul de înclinare este calculat cu un filtru complementar din datele de accelerometru și giroscop.
- Motoarele sunt controlate prin PWM bidirecțional, iar sistemul reacționează la obstacole apropiate (<20 cm) prin inversarea direcției.
- Un LED (pin 13) indică funcționarea periodică (1 Hz).

<note tip>

#include <Wire.h> #include <I2Cdev.h> #include <MPU6050.h> #include <math.h> #include <NewPing.h>

#define leftMotorPWMPin 6 #define leftMotorDirPin 7 #define rightMotorPWMPin 5 #define rightMotorDirPin 4 #define TRIGGER_PIN 9 #define ECHO_PIN 8 #define MAX_DISTANCE 75 #define Kp 40 #define Kd 0.05 #define Ki 40 #define sampleTime 0.005 #define targetAngle -2.5

MPU6050 mpu; NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

int16_t accY, accZ, gyroX; volatile int motorPower, gyroRate; volatile float accAngle, gyroAngle, currentAngle, prevAngle = 0; volatile float error, prevError = 0, errorSum = 0; volatile byte count = 0; int distanceCm;

void setMotors(int leftMotorSpeed, int rightMotorSpeed) {

```
  if (leftMotorSpeed >= 0) {
      analogWrite(leftMotorPWMPin, leftMotorSpeed);
      digitalWrite(leftMotorDirPin, LOW);
```

```
  } else {
      analogWrite(leftMotorPWMPin, 255 + leftMotorSpeed);
      digitalWrite(leftMotorDirPin, HIGH);
  }
```

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

```
}
```

void init_PID() {

```
  cli(); // disable global interrupts
  TCCR1A = 0;
  TCCR1B = 0;
  OCR1A = 9999;
  TCCR1B |= (1 << WGM12); // CTC mode
  TCCR1B |= (1 << CS11);  // prescaler 8
  TIMSK1 |= (1 << OCIE1A); // enable timer interrupt
  sei(); // enable global interrupts
```

```
}
```

void setup() {

```
  pinMode(leftMotorPWMPin, OUTPUT);
  pinMode(leftMotorDirPin, OUTPUT);
  pinMode(rightMotorPWMPin, OUTPUT);
  pinMode(rightMotorDirPin, OUTPUT);
  pinMode(13, OUTPUT);
```

```
  Wire.begin();
  mpu.initialize();
```

```
  mpu.setYAccelOffset(1593);
  mpu.setZAccelOffset(963);
  mpu.setXGyroOffset(40);
```

```
  init_PID();
```

```
}
```

void loop() {

```
  accY = mpu.getAccelerationY();
  accZ = mpu.getAccelerationZ();
```

```
  gyroX = mpu.getRotationX();

  motorPower = constrain(motorPower, -255, 255);
  setMotors(motorPower, motorPower);

  if ((count % 20) == 0) {
      distanceCm = sonar.ping_cm();
  }

  if ((distanceCm < 20) && (distanceCm != 0)) {
      setMotors(-motorPower, motorPower);
  }

}

ISR(TIMER1_COMPA_vect) {

  accAngle = atan2(accY, accZ) * RAD_TO_DEG;
  gyroRate = map(gyroX, -32768, 32767, -250, 250);
  gyroAngle = (float)gyroRate * sampleTime;
  currentAngle = 0.9934 * (prevAngle + gyroAngle) + 0.0066 * accAngle;

  error = currentAngle - targetAngle;
  errorSum += error;
  errorSum = constrain(errorSum, -300, 300);

  motorPower = Kp * error + Ki * errorSum * sampleTime - Kd * (currentAngle
- prevAngle) / sampleTime;
  prevAngle = currentAngle;

  count++;
  if (count == 200) {
      count = 0;
      digitalWrite(13, !digitalRead(13));
  }

}
```