Rover MEGA PROJECT

RoboTech

- **PWM from Raspberry:** Implementation and control of Pulse Width Modulation (PWM) signals using a Raspberry Pi for precise motor control.
- **Switching Circuit:** Design and analysis of an efficient switching circuit to manage high-power loads.
- **DC:** Exploration of DC motor characteristics and performance under various operating conditions.
- **IMU:** Integration of an Inertial Measurement Unit (IMU) for real-time orientation and motion tracking.
- **SPI:** Configuration of the Serial Peripheral Interface (SPI) for fast and reliable communication between devices.
- **ROS bridge:** Development of a ROS bridge to enable seamless communication between robotic hardware and software.

Introduction

This is a brief of our research on each topic concerning the rover. It will include the purpose and functionality of each topic along its relevance to the rover.

1. PWM from Raspberry

TITLE:

- item 1.
- item 2.
- item 3.
- item 4.

2. Switching Circuit

TITLE:

- item 1.
- item 2.
- item 3.
- item 4.

3. DC Motors

TITLE:

- item 1.
- item 2.
- item 3.
- item 4.

4. Inertial Measurement Unit (IMU)

4.1. Introduction

An Inertial Measurement Unit (IMU) is a critical sensor in rover systems, providing essential data for navigation, control, and stability here are the main uses of the IMU in the rover:

- **Orientation and Attitude Monitoring:** The IMU provides real-time data on the rover's roll, pitch, and yaw, which represent its orientation in 3D space. This information is crucial for maintaining stability, especially when traversing uneven terrains or slopes.
- **Navigation:** Especially path correction, the IMU's gyroscope data is used to detect deviations from the intended trajectory, allowing the rover to adjust its movement and stay on course.
- **Obstacle Avoidance and Terrain Adaptation:** The IMU helps the rover detect and adapt to sudden changes in terrain, such as inclines or declines, by adjusting speed and direction accordingly. It works in conjunction with other sensors like ultrasonic sensors or cameras to navigate around obstacles.

4.2. IMU Configuration and Components

Mostly we only use the accelerometer (to measure the acceleration vector) and the gyroscope (to measure angular velocity), the temperature sensor is not really used. Here are the details of the IMU that we will use in our rover:

- **Module:** MPU6050 (6-axis, accelerometer + gyroscope).

- **Arduino IDE Library:** Adafruit_MPU6050.
- **Communication with the ESP32:** I2C, using "Wire.h" library.

4.3. Test Code

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

void setup() {
  Serial.begin(115200);  // Initialize serial communication
  Wire.begin();          // Initialize I2C communication

  // Initialize MPU6050
  mpu.initialize();
  if (mpu.testConnection()) {
    Serial.println("MPU6050 connection successful!");
  } else {
    Serial.println("MPU6050 connection failed!");
    while (1);  // Stop execution if IMU fails
  }
}

void loop() {
  // Read accelerometer and gyroscope values
  int16_t ax, ay, az, gx, gy, gz;
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

  // Print data to Serial Monitor
  Serial.print("Ax: "); Serial.print(ax);
  Serial.print(" Ay: "); Serial.print(ay);
  Serial.print(" Az: "); Serial.print(az);
  Serial.print(" | Gx: "); Serial.print(gx);
  Serial.print(" Gy: "); Serial.print(gy);
  Serial.print(" Gz: "); Serial.println(gz);

  delay(500);  // Wait for 500ms before the next reading
}
```

5. Serial Peripheral Interface (SPI)

5.1. Introduction

Serial Peripheral Interface (SPI) is a synchronous serial communication protocol used to transfer data between a master device and one or more peripheral devices. It operates by shifting bits one at a time through a full-duplex connection, with a clock signal synchronizing the data transmission.

SPI typically uses four main lines:

- **MOSI (Master Out Slave In):** Transmits data from the master to the slave.
- **MISO (Master In Slave Out):** Transmits data from the slave to the master.
- **SCLK (Serial Clock):** Provides a clock signal for synchronization.
- **CS/SS (Chip Select/Slave Select):** Used to select which peripheral device is being communicated with at any given time.

5.2. SPI in ESP32

The ESP32 microcontroller supports SPI communication on several GPIO pins, which can be configured as per the requirements of the application, in our case the communication of the ESP32 with the Raspberry Pi. The ESP32 has hardware SPI support, which provides efficient data transfer with minimal CPU involvement. This makes it ideal for high-speed applications.

Key Features:

- **Multiple SPI Buses:** The ESP32 has three SPI hardware peripherals (SPI0, SPI1, SPI2) that can be used independently, allowing simultaneous communication with multiple peripherals.
- **Flexible Pin Configuration:** The ESP32 allows the user to assign different pins to the SPI interface, which provides flexibility in designing circuits.
- **DMA Support:** The ESP32 supports Direct Memory Access (DMA) for SPI communication, which can offload data transfer to a peripheral and improve performance by reducing CPU usage.

5.2.1. Test Code

```
#include <SPI.h>

void setup() {
  // Set SPI settings: SPI clock, bit order, data mode
  SPI.begin();  // Start SPI bus

  // Set the CS pin as output
  pinMode(SS, OUTPUT);
}

void loop() {
```

```
  // Initiate communication with the SPI device
  digitalWrite(SS, LOW);  // Select the SPI device
  SPI.transfer(0xFF);      // Send data
  digitalWrite(SS, HIGH);  // Deselect the SPI device

  delay(1000);  // Wait for a second before next transmission
}
```

5.3. SPI in Raspberry Pi

5.3.1. Test Code

```
import spidev
import time

# Create an SPI object
spi = spidev.SpiDev()
spi.open(0, 0)  # Open SPI bus 0, chip select 0
spi.max_speed_hz = 50000  # Set SPI speed

while True:
    response = spi.xfer2([0xFF])  # Send data and receive response
    print("Received:", response)
    time.sleep(1)
```

6. ROS bridge

TITLE:

- item 1.
- item 2.
- item 3.
- item 4.

Conclusion

CONCLUSION