Lab report 5 of CSE461

Submitted By:

Group 4

Name	ID
Tahmid Chowdhury	19201115
Tabassum Nusrat Jahan	19201027
Sadman Sakib Nahid	19201029
Mahfuza Sultana Mim	18101703
Fairuz Anika	20301464

1. Name of the experiment: Interfacing an IMU with Raspberry Pi.

2. Objective: Interfacing an IMU (Inertial Measurement Unit) with a Raspberry Pi experiment where an IMU typically includes sensors like accelerometers, gyroscopes, and sometimes magnetometers, which allow you to measure motion, orientation, and environmental factors. In this experiment, we'll use an MPU-6050 IMU module, which combines accelerometer and gyroscope sensors.

3. Equipment:

- ➢ Raspberry Pi 4
- ➢ Connecting wires
- ➢ Breadbroad
- ➢ MPU-6050 IMU module
- ➢ Internet connection (for installing libraries if needed)

4. Experimental Setup: In this experiment, we interface an IMU with Raspberry pi. To do this firstly we have connected the IMU in a breadboard. Then the VCC of the IMU connected with the Raspberry-pi board pin number 1 which can provide 3.3 volt with the help of a jumper wire. Then we connected the ground with a ground pin of the Raspberry-pi. After these We have connected SCL and ECL together and those two connected with the GPIO3 of Raspberry-pi which is basically an I2C1 SCL port. Then we have connected SDL and EDL together and also connected those two with the GPIO2 of Raspberry-pi which is an I2C1 SDA port of Raspberry pi. After doing all of this task we have run some codes in the terminal and then run the code. Our work ran successfully.

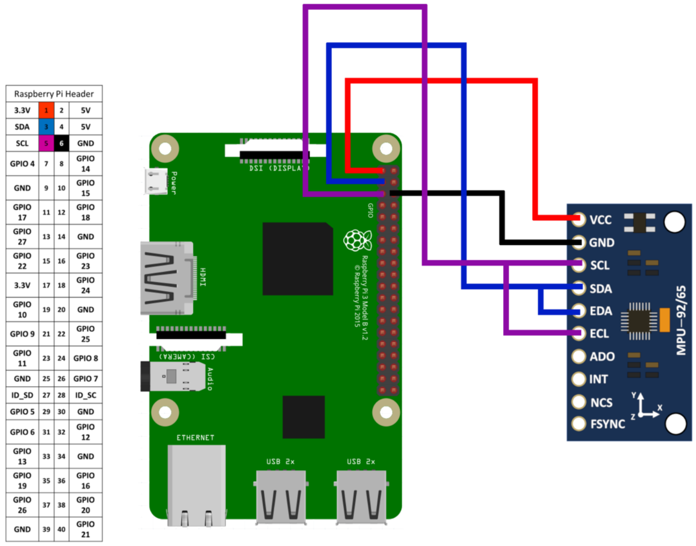


Fig1: circuit diagram

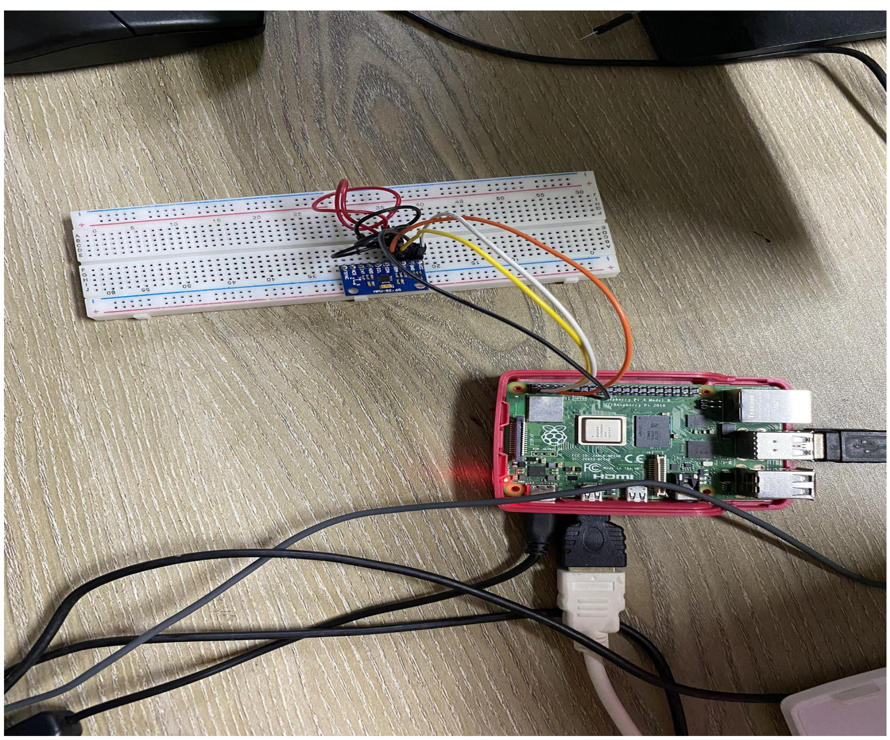


Fig 2: Circuit Implementation

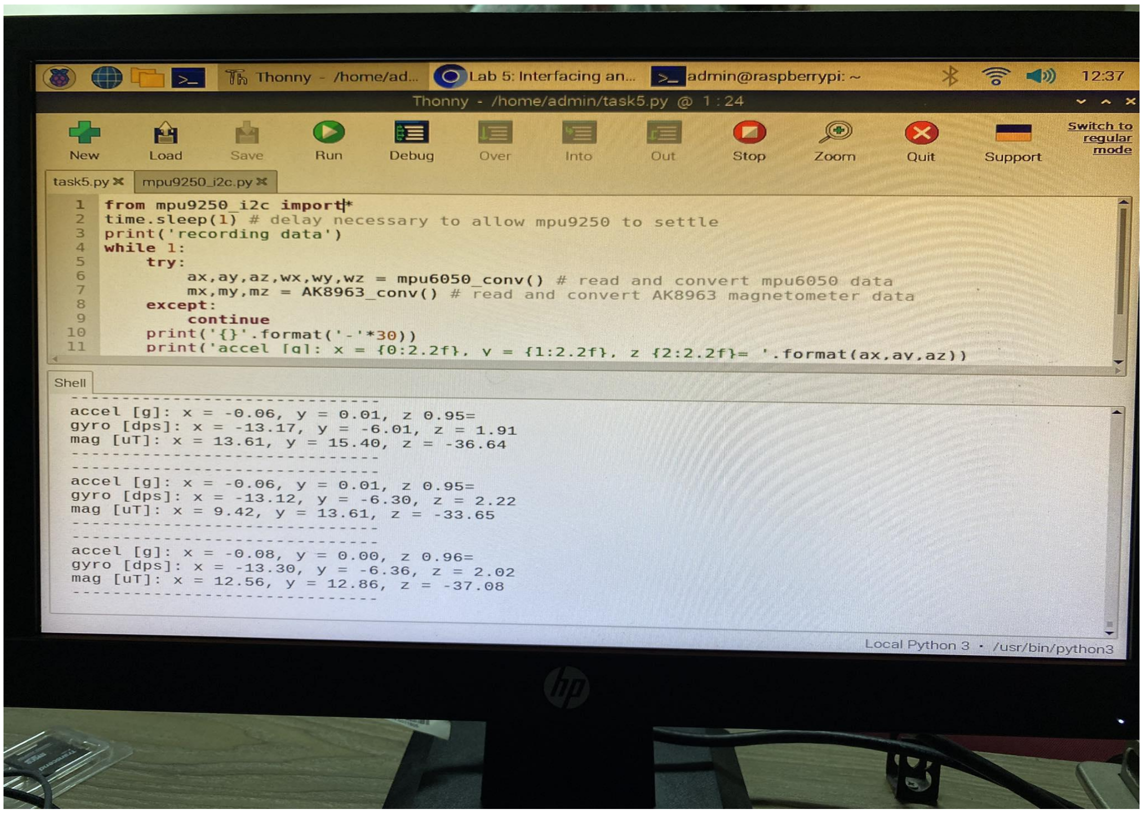
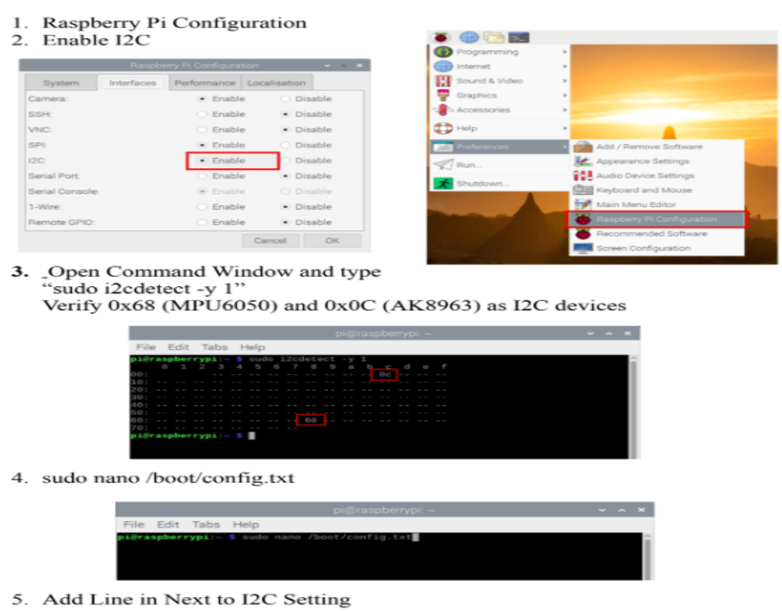
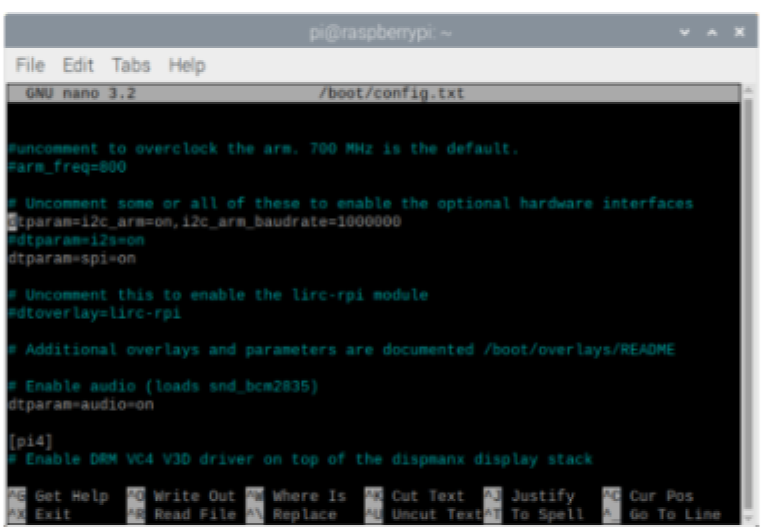


Fig 3: Code ran Successfully

5. Code:

Setup Configuration with terminal:





Library part:

```
# this is to be saved in the local folder under the name "mpu9250_i2c.py"
# it will be used as the I2C controller and function harbor for the project
# refer to datasheet and register map for full explanation

import smbus,time

def MPU6050_start():
    # alter sample rate (stability)
    samp_rate_div = 0 # sample rate = 8 kHz/(1+samp_rate_div)
    bus.write_byte_data(MPU6050_ADDR, SMPLRT_DIV, samp_rate_div)
    time.sleep(0.1)
    # reset all sensors
    bus.write_byte_data(MPU6050_ADDR,PWR_MGMT_1,0x00)
    time.sleep(0.1)
    # power management and crystal settings
    bus.write_byte_data(MPU6050_ADDR, PWR_MGMT_1, 0x01)
    time.sleep(0.1)
    #Write to Configuration register
    bus.write_byte_data(MPU6050_ADDR, CONFIG, 0)
    time.sleep(0.1)
    #Write to Gyro configuration register
    gyro_config_sel = [0b00000,0b010000,0b10000,0b11000] # byte registers
    gyro_config_vals = [250.0,500.0,1000.0,2000.0] # degrees/sec
```

```
    gyro_indx = 0
    bus.write_byte_data(MPU6050_ADDR, GYRO_CONFIG, int(gyro_config_sel[gyro_indx]))
    time.sleep(0.1)
    #Write to Accel configuration register
    accel_config_sel = [0b00000,0b01000,0b10000,0b11000] # byte registers
    accel_config_vals = [2.0,4.0,8.0,16.0] # g (g = 9.81 m/s^2)
    accel_indx = 0
    bus.write_byte_data(MPU6050_ADDR, ACCEL_CONFIG,int(accel_config_sel[accel_indx]))
    time.sleep(0.1)
    # interrupt register (related to overflow of data [FIFO])
    bus.write_byte_data(MPU6050_ADDR, INT_ENABLE, 1)
    time.sleep(0.1)
    return gyro_config_vals[gyro_indx],accel_config_vals[accel_indx]
def read_raw_bits(register):
    # read accel and gyro values
    high = bus.read_byte_data(MPU6050_ADDR, register)
    low = bus.read_byte_data(MPU6050_ADDR, register+1)
    # combine higha and low for unsigned bit value
    value = ((high << 8) | low)
    # convert to +- value
    if(value > 32768):
        value -= 65536
    return value
def mpu6050_conv():
    # raw acceleration bits
    acc_x = read_raw_bits(ACCEL_XOUT_H)
```

```
    acc_y = read_raw_bits(ACCEL_YOUT_H)
    acc_z = read_raw_bits(ACCEL_ZOUT_H)
    # raw temp bits
    ## t_val = read_raw_bits(TEMP_OUT_H) # uncomment to read temp
    # raw gyroscope bits
    gyro_x = read_raw_bits(GYRO_XOUT_H)
    gyro_y = read_raw_bits(GYRO_YOUT_H)
    gyro_z = read_raw_bits(GYRO_ZOUT_H)
    #convert to acceleration in g and gyro dps
    a_x = (acc_x/(2.0**15.0))*accel_sens
    a_y = (acc_y/(2.0**15.0))*accel_sens
    a_z = (acc_z/(2.0**15.0))*accel_sens
    w_x = (gyro_x/(2.0**15.0))*gyro_sens
    w_y = (gyro_y/(2.0**15.0))*gyro_sens
    w_z = (gyro_z/(2.0**15.0))*gyro_sens
    ## temp = ((t_val)/333.87)+21.0 # uncomment and add below in return
    return a_x,a_y,a_z,w_x,w_y,w_z
def AK8963_start():
    bus.write_byte_data(AK8963_ADDR,AK8963_CNTL,0x00)
    time.sleep(0.1)
    AK8963_bit_res = 0b0001 # 0b0001 = 16-bit
    AK8963_samp_rate = 0b0110 # 0b0010 = 8 Hz, 0b0110 = 100 Hz
    AK8963_mode = (AK8963_bit_res <<4)+AK8963_samp_rate # bit conversion
    bus.write_byte_data(AK8963_ADDR,AK8963_CNTL,AK8963_mode)
    time.sleep(0.1)
def AK8963_reader(register):
```

```
    # read magnetometer values
    low = bus.read_byte_data(AK8963_ADDR, register-1)
    high = bus.read_byte_data(AK8963_ADDR, register)
    # combine higha and low for unsigned bit value
    value = ((high << 8) | low)
    # convert to +- value
    if(value > 32768):
        value -= 65536
    return value

def AK8963_conv():
    # raw magnetometer bits
    loop_count = 0
    while 1:
        mag_x = AK8963_reader(HXH)
        mag_y = AK8963_reader(HYH)
        mag_z = AK8963_reader(HZH)
    # the next line is needed for AK8963
        if bin(bus.read_byte_data(AK8963_ADDR,AK8963_ST2))=='0b10000':

            break
        loop_count+=1
    #convert to acceleration in g and gyro dps
    m_x = (mag_x/(2.0**15.0))*mag_sens
    m_y = (mag_y/(2.0**15.0))*mag_sens
    m_z = (mag_z/(2.0**15.0))*mag_sens
```

```
    return m_x,m_y,m_z
# MPU6050 Registers
MPU6050_ADDR = 0x68
PWR_MGMT_1 = 0x6B
SMPLRT_DIV = 0x19
CONFIG = 0x1A
GYRO_CONFIG = 0x1B
ACCEL_CONFIG = 0x1C
INT_ENABLE = 0x38
ACCEL_XOUT_H = 0x3B
ACCEL_YOUT_H = 0x3D
ACCEL_ZOUT_H = 0x3F
TEMP_OUT_H = 0x41
GYRO_XOUT_H = 0x43
GYRO_YOUT_H = 0x45
GYRO_ZOUT_H = 0x47
#AK8963 registers
AK8963_ADDR = 0x0C
AK8963_ST1 = 0x02
HXH = 0x04
HYH = 0x06
HZH = 0x08
AK8963_ST2 = 0x09
AK8963_CNTL = 0x0A
mag_sens = 4900.0 # magnetometer sensitivity: 4800 uT
# start I2C driver
```

```
bus = smbus.SMBus(1) # start comm with i2c bus
gyro_sens,accel_sens = MPU6050_start() # instantiate gyro/accel
AK8963_start() # instantiate magnetometer
```

Task1 (code):

```
from mpu9250_i2c import*
time.sleep(1) # delay necessary to allow mpu9250 to settle
print('recording data')
while 1:
    try:
        ax,ay,az,wx,wy,wz = mpu6050_conv() # read and convert mpu6050 data
        mx,my,mz = AK8963_conv() # read and convert AK8963 magnetometer data
    except:
        continue
    print('{}'.format('-'*30))
    print('accel [g]: x = {0:2.2f}, y = {1:2.2f}, z {2:2.2f}= '.format(ax,ay,az))
    print('gyro [dps]: x = {0:2.2f}, y = {1:2.2f}, z = {2:2.2f}'.format(wx,wy,wz))
    print('mag [uT]: x = {0:2.2f}, y = {1:2.2f}, z = {2:2.2f}'.format(mx,my,mz))
    print('{}'.format('-'*30))
```

- time.sleep(1)

6. Result:

The stated objective of connecting a Raspberry Pi and an MPU-6050 IMU module was successfully accomplished. The Raspberry Pi was successfully combined with the IMU module, which combines accelerometer and gyroscope sensors, to assess motion, orientation, and the surroundings. Data from the IMU sensors was precisely recorded and processed by the Raspberry Pi throughout the experiment. The calculation of orientation changes was made possible by the accelerometer data, which supplied real-time information.

7. Discussion:

In order to interface an IMU with a Raspberry Pi, the IMU sensor must be connected to one of the device's GPIO pins or an interface like I2C or SPI. Choosing an IMU, wiring it, installing necessary libraries, reading data from the sensor using Python programming, calibrating the IMU, processing the data, and utilizing it in our project are the steps. For accuracy and reliability, thorough testing is necessary.