THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY
ISDN 2602

Laboratory 2: Sensors (5%)

Github classroom link
https://classroom.github.com/a/67yXOYYA
Please submit your code before: 23:59   01/ Oct/ 2025

A) Objectives

- To get familiar with ESP32 and Arduino IDE
- To get familiar with different sensors such as ultrasonic sensor and inertial measurement Unit (IMU).
- To understand the processing of the raw signals and sensor fusion.

B) <u>Tasks</u>

In this lab, we will use the ultrasonic sensor and IMU on the car, which will also be used in the final project. The car is shown below:
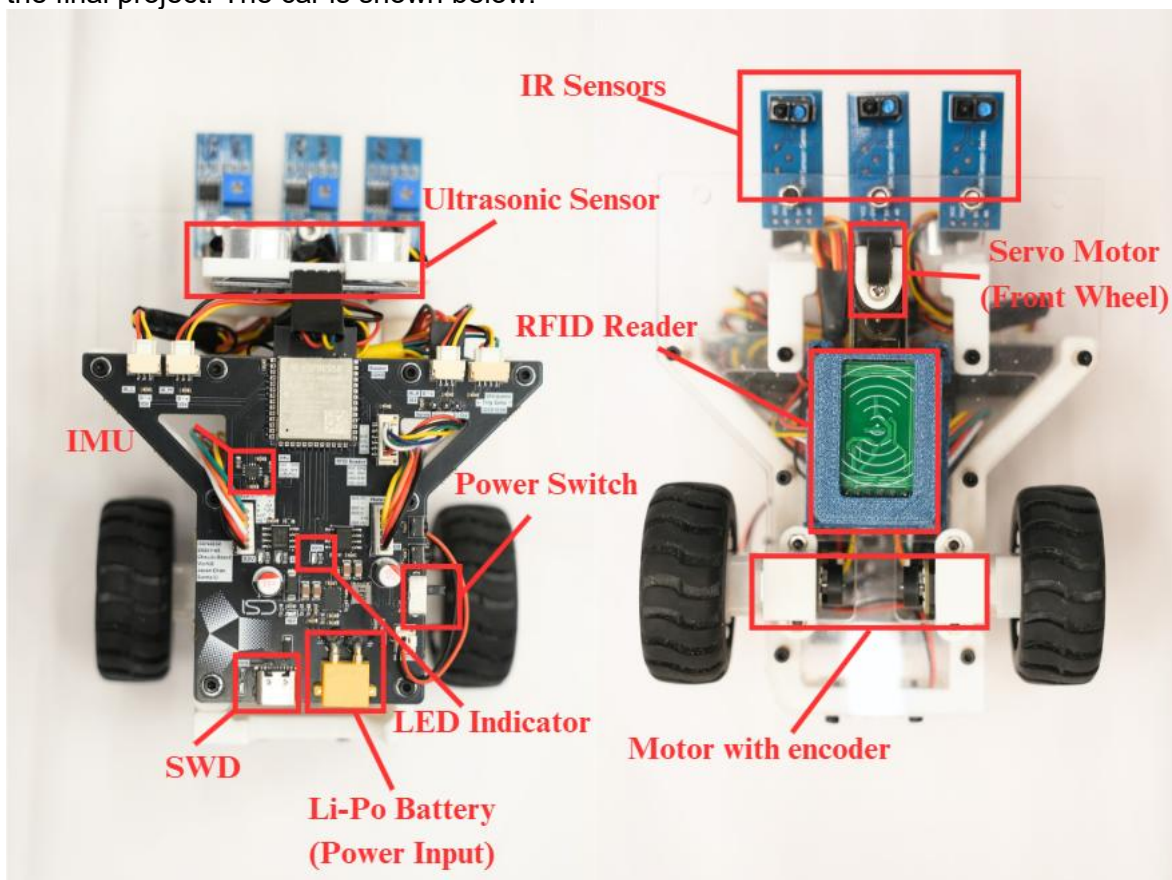
Composition of the robotic car

Material for Task 1 & 2

Setting in Arduino IDE

Since the development board on the robotic car does not have a pre-set environment in Arduino default board library. Please follow the Upload Setting below. ~o~Otherwise the serial port and flash in MCU may not work properly.

1. Choose board as "ESP32S3 Dev Module".

2. Then go to "Tools" and make other setting as show below:

USB CDC On Boot: "Enabled"   ▸
CPU Frequency: "240MHz (WiFi)"   ▸
Core Debug Level: "None"   ▸
USB DFU On Boot: "Disabled"   ▸
Erase All Flash Before Sketch Upload: "Disabled"   ▸
Events Run On: "Core 1"   ▸
Flash Mode: "QIO 80MHz"   ▸
Flash Size: "8MB (64Mb)"   ▸
JTAG Adapter: "Disabled"   ▸
Arduino Runs On: "Core 1"   ▸
USB Firmware MSC On Boot: "Disabled"   ▸
Partition Scheme: "No OTA (2MB APP/2MB SPIFFS)"   ▸
PSRAM: "Disabled"   ▸
Upload Mode: "USB-OTG CDC (TinyUSB)"   ▸
Upload Speed: "921600"   ▸
USB Mode: "Hardware CDC and JTAG"   ▸
Zigbee Mode: "Disabled"   ▸

Task 1: Ultrasonic Sensor.

In this task, we will use the robotic car as the platform to test the routing algorithm. An HC-SR04 ultrasonic sensor is used in this task.
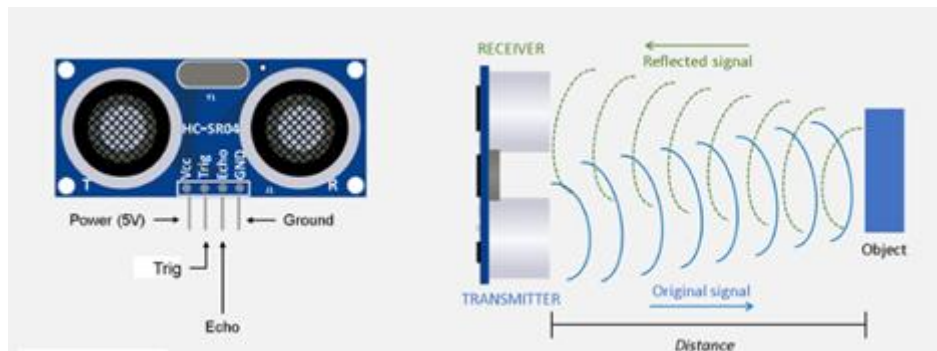

Figure 4: Work Principle of ultrasonic sensor

Introduction of ultrasonic sensor

An ultrasonic sensor operates by emitting high-frequency sound waves and measuring the time it takes for the waves to bounce back after hitting an object. Here's a brief workflow of how it functions:

1. Transmission: The sensor's transmitter emits ultrasonic sound waves, typically at frequencies above 20 kHz, which are inaudible to humans.
2. Propagation: These sound waves travel through the air towards the target object.
3. Reflection: Upon encountering an object, the sound waves are reflected back to the sensor.
4. Reception: The sensor's receiver detects the reflected sound waves (echo).
5. Time Measurement: The sensor calculates the time interval between the emission of the sound wave and the reception of the echo.
6. Distance Calculation: Using the speed of sound in air (approximately 343 meters per second at room temperature), the sensor computes the distance to the object using the formula:

$$Distance = \frac{Speed\ of\ sound\ \times Time\ interval}{2}$$

7. Output: The sensor outputs the calculated distance, which can be used by a microcontroller or other processing unit for further actions, such as obstacle avoidance, object detection, or distance measurement.

Task to do:

1. Open the "Task_1.ino" file with Arduino IDE and define the speed of sound

- The pinout of the PCB for the ultrasonic sensor is shown as follows:
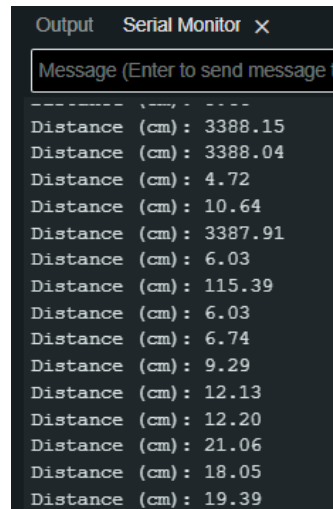
```
#define trigPin 39
#define echoPin 38
```

- Input the speed of sound in terms of m/s

```
/*define sound speed in m*/
#define SOUND_SPEED
```

- Modify the equation:

```
// Calculate the distance (in m)
distance = (duration * SOUND_SPEED)/2;
```

2. Upload the code to the development board.

3. Check the serial monitor and test whether the distance is correct or not


```
Output    Serial Monitor  ✕

Message (Enter to send message t

Distance (cm): 3388.15
Distance (cm): 3388.04
Distance (cm): 4.72
Distance (cm): 10.64
Distance (cm): 3387.91
Distance (cm): 6.03
Distance (cm): 115.39
Distance (cm): 6.03
Distance (cm): 6.74
Distance (cm): 9.29
Distance (cm): 12.13
Distance (cm): 12.20
Distance (cm): 21.06
Distance (cm): 18.05
Distance (cm): 19.39
```

4. Find three distances or dimensions of objects to measure and record the value in the answer sheet.

	The objects you measured	Value
1	Phone directly in front of ultrasonic sensor	1.97cm
2	Phone placed in front of infrared sensor	2.60cm
3	Phone placed at edge of table	39.66cm

Check Point:
**Commit the code to the GitHub classroom. Show your result to the TA.**

Task 2:Inertial Measurement Unit (IMU) and Visualization.

The Inertial Measurement Unit, or IMU, is a critical component in modern navigation and motion tracking systems. It typically consists of accelerometers and gyroscopes, and can measure an object's specific force, angular rate, and orientation.

An IMU typically consists of:
- **Accelerometers**: Measure linear acceleration a in m/s².
- **Gyroscopes**: Measure angular velocity w in degrees/sec.
- **Magnetometers** (optional): Measure the magnetic field strength m in uT (micro Tesla) or Gauss.

To track orientation
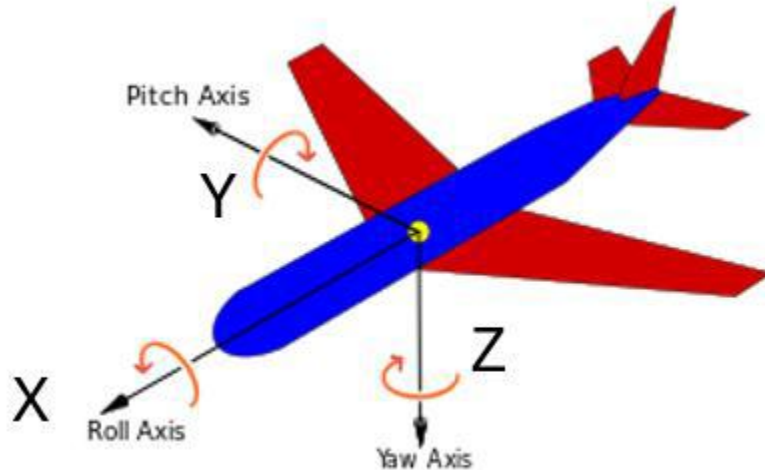
- Using Gyro to estimate orientation (e.g., roll, pitch, yaw), you integrate the angular velocity over time:

$$\theta(t) = \theta_0 + \int_0^t \omega(\tau)d\tau$$

- Using accelerometer to estimate orientation due to gravity (about 9.81 m/s², pointing toward the center of the Earth). When the device is stationary or moving at a constant velocity, the main acceleration it measures is gravity. By analyzing the gravity components along each axis, you can estimate the device's tilt angles relative to the horizontal plane—specifically, the roll and pitch angles.

$$Roll = \phi = \arctan\left(\frac{az}{ay}\right)$$

$$Pitch = \theta = arctan\left(-\frac{ax}{ay^2 + az^2}\right)$$

Sources of Error in Orientation Estimation

Sensor	Strengths	Error
Gyroscope	Fast, smooth, no noise	Drift over time
Accelerometer	Stable long-term, no drift	Noisy, affected by motion

How to reduce the effect of error

A. Low-Pass Filter

- **Purpose:** Removes high-frequency noise from accelerometer data.
- **How it helps:** Since gravity is a low-frequency (constant) signal, a low-pass filter can help isolate the gravity component from rapid movements or vibrations.
- **Limitation:** If you filter too much, you may introduce lag or miss rapid orientation changes.

B. Complementary Filter

- **Purpose:** Combines the strengths of both accelerometer and gyroscope data.
- **Gyroscope:** Good for short-term, fast changes (high-frequency), but drifts over time.
- **Accelerometer:** Good for long-term, stable orientation (low-frequency), but noisy and affected by motion.
- **How it works:** The complementary filter blends the two:
  - Uses the gyroscope for short-term changes.
  - Uses the accelerometer for long-term correction.

The IMU model used in this lab is ICM-48826P.

ICM-48826P & XYZ axis orientation

In this task, we will use Low-Pass Filter and Complementary Filter to obtain more accurate data from the IMU. The sensing output with and without data fusion will be uploaded to a web app for visualization and comparison purposes.

Skeleton Code

- Initialization of ICM42688P (I2C)

```cpp
/*------------ DO NOT Change the code BELOW ------------*/
// an ICM42688 object with the ICM42688 sensor on I2C bus with address 0x68 using SDA pin 17 and SCL pin 18
ICM42688 IMU(Wire, 0x68, IMU_SDA, IMU_SCL);

void setup() {
  // serial to display data
  Serial.begin(115200);
  while (!Serial) {}

  // start communication with IMU
  int status = IMU.begin();
  if (status < 0) {
    Serial.println("IMU initialization unsuccessful");
    Serial.println("Check IMU wiring or try cycling power");
    Serial.print("Status: ");
    Serial.println(status);
    while (1) {}
  }

  // setting the accelerometer full scale range to +/-8G
  IMU.setAccelFS(ICM42688::gpm8);
  // setting the gyroscope full scale range to +/-500 deg/s
  IMU.setGyroFS(ICM42688::dps500);

  // set output data rate to 12.5 Hz
  IMU.setAccelODR(ICM42688::odr12_5);
  IMU.setGyroODR(ICM42688::odr12_5);

  Serial.println("---IMU Initialized---");
}
/*------------ DO NOT Change the code ABOVE ------------*/
```

- Low-Pass Filter and Filter parameters:

```
/*Define constant */
const float LowPassFilterAlpha = 0.3f  ; //*** Set filter constant for accelerometer which Alpha + Beta = 1.0f ***
const float LowPassFilterBeta =  0.7f ;

/*Applying Filter*/
filteredAccX = LowPassFilterAlpha * (IMU.accX()) + (1 - LowPassFilterAlpha) * filteredAccX;
filteredAccY = LowPassFilterAlpha * (IMU.accY()) + (1 - LowPassFilterAlpha) * filteredAccY;
filteredAccZ = LowPassFilterAlpha * (IMU.accZ()) + (1 - LowPassFilterAlpha) * filteredAccZ;
```

- Filter function:

```
// Apply low-pass filter to gyr)oscope readings
filteredGyroX = LowPassFilterBeta * (IMU.gyrX()) * DEG_TO_RAD + (1 - LowPassFilterBeta) * filteredGyroX;
filteredGyroY = LowPassFilterBeta * (IMU.gyrY()) * DEG_TO_RAD + (1 - LowPassFilterBeta) * filteredGyroY;
filteredGyroZ = LowPassFilterBeta * (IMU.gyrZ()) * DEG_TO_RAD + (1 - LowPassFilterBeta) * filteredGyroZ;

// Calculate roll and pitch from accelerometer data
float acc_roll = atan2(filteredAccY, sqrt(filteredAccX * filteredAccX + filteredAccZ * filteredAccZ));
float acc_pitch = atan2(-filteredAccX, sqrt(filteredAccY * filteredAccY + filteredAccZ * filteredAccZ));

// Integrate gyroscope data to get roll, pitch, and yaw
float gyro_roll = roll + filteredGyroX * dt;
float gyro_pitch = pitch + filteredGyroY * dt;
float gyro_yaw = yaw + filteredGyroZ * dt;

// Apply complementary filter
roll = ComplementaryFilterALPHA * gyro_roll + (1 - ComplementaryFilterALPHA) * acc_roll;
pitch = ComplementaryFilterALPHA * gyro_pitch + (1 - ComplementaryFilterALPHA) * acc_pitch;
yaw = gyro_yaw;  // Yaw is not corrected by accelerometer
```

- Display mode:

```
/*------------ DO NOT Change the code ABOVE ------------*/
/*Enable Filter*/
bool Filter = true;              //True:  Enable
                                 //False: Disale

/*Enable Serial Plot*/
bool SerialPlotGrapgh = false; //True:  Enable
                                 //False: Disale

/*------------ DO NOT Change the code BELOW ------------*/
```

Task to do:

1. Open the "**Task_2.ino**" file in Task 2 folder with Arduino IDE.
2. Change the code to disable the Serial Plot and the filter.
3. Upload the "**Task_2.ino**" from Arduino IDE to the development board
4. Open the Chrome browser, then visit "https://imu.isdn2602.site". Click "Open Port" and choose the port connected to the development board. You should be able to see the sensor values and a mapped object, "ISDN 2602".
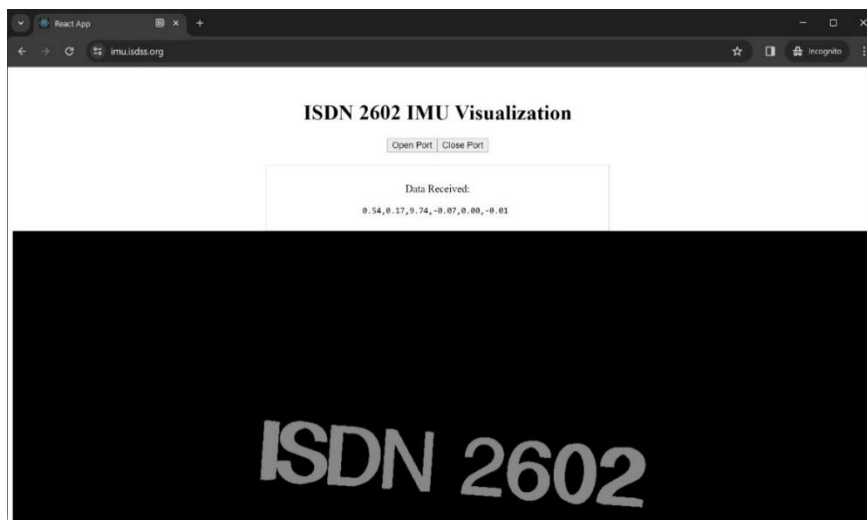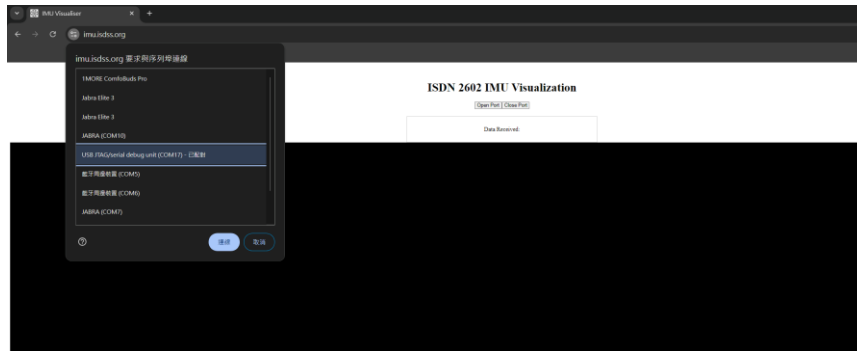
5. Try to move and rotate the robotic car. Observe the change of values and the movement ?? of the object.
6. Then change the code to only activiate the Low-Pass Filter and observe the change of the movement.
7. Then change the code to activate both Low-Pass Filter and Complementary Filter and see the result.
8. Enable the Serial Plot in the code and plot the IMU data by Serial Plotter. Repeat Steps 6 and 7.

__Check Point:__

__Commit the code to the GitHub classroom. Show your result to the TA.__

--End--

Appendix

1. Include Library and Define Pins

```
#include "Arduino.h"
```

- #include "Arduino.h": This includes the Arduino library, allowing us to use Arduino functions.

```
#define trigPin 16
#define echoPin 15
```

- Defines the trigger pin as pin 16.
- Defines the echo pin as pin 15.

2. Define Sound Speed

```
#define SOUND_SPEED
```

- #define SOUND_SPEED 001: Defines the speed of sound in air (in meters). This value should be 340 meters/second.

3. Define Variables

```
long duration;
float distance;
```

- long duration;: This variable will store the time it takes for the sound wave to travel to the obstacle and back (in microseconds).
- float distance;: This variable will store the calculated distance (in meters).

4. Setup Function

```
Serial.begin(115200)
```

- Initializes serial communication at a baud rate of 115200.

```
pinMode(trigPin, OUTPUT)
```

- Sets the trigger pin as an output.

```
pinMode(echoPin, INPUT)
```

- Sets the echo pin as an input.

```
Serial.println("Ultrasonic Sensor is set")
```

- Prints a message to the serial monitor indicating that the sensor is set.

```
delay(10);
```

- Pauses for 10 milliseconds.

5. Main Loop

```
digitalWrite(trigPin, LOW)
```

- Sets the trigger pin to low, clearing any previous signal.

```
delayMicroseconds(2)
```

- Waits for 2 microseconds.

```
digitalWrite(trigPin, HIGH)
```

- Sets the trigger pin to high, sending out an ultrasonic signal.

```
delayMicroseconds(10)
```

- Keeps the pin high for 10 microseconds.

```
digitalWrite(trigPin, LOW)
```

- Sets the trigger pin back to low, ending the signal.

```
duration = pulseIn(echoPin, HIGH)
```

- Reads the duration of the echo pin being high, which gives the time taken for the sound wave to return (in microseconds).

```
distance = (duration * SOUND_SPEED/100)/2
```

- Calculates the distance. The formula is: Distance = (Time × Speed of Sound) / 2. We divide by 2 because the sound wave travels to the obstacle and back.

```
Serial.print("Distance (cm): ")
Serial.println(distance/100)
```

- Prints the calculated distance (converted to centimeters).

```
delay(1000)
```

- Pauses for 1 second before the next measurement.

Code implementation for Low Pass Filter

For ICM42688-P, there are TWO options of protocol available which are SPI and I2C. In this task, we will use I2C for the communication for simplicity, the following code is the initialization of the I2C channel and configuration of IMU.

Once the communication between MCU and IMU is established, we can use the following function to get all the data from IMU and stored them in MCU for further processing.

```cpp
void loop() {
  // read the sensor
  IMU.getAGT();
```

The following code is applying Low Pass Filter in C++. Which there are Alpha and Beta (1-Alpha). Noted that the sum of Alpha and Beta should equal to 1.

```cpp
filteredAccX = LowPassFilterAlpha * (IMU.accX()) + (1 - LowPassFilterAlpha) * filteredAccX;
filteredAccY = LowPassFilterAlpha * (IMU.accY()) + (1 - LowPassFilterAlpha) * filteredAccY;
filteredAccZ = LowPassFilterAlpha * (IMU.accZ()) + (1 - LowPassFilterAlpha) * filteredAccZ;

// Apply low-pass filter to gyr)oscope readings
filteredGyroX = LowPassFilterBeta * (IMU.gyrX()) * DEG_TO_RAD + (1 - LowPassFilterBeta) * filteredGyroX;
filteredGyroY = LowPassFilterBeta * (IMU.gyrY()) * DEG_TO_RAD + (1 - LowPassFilterBeta) * filteredGyroY;
filteredGyroZ = LowPassFilterBeta * (IMU.gyrZ()) * DEG_TO_RAD + (1 - LowPassFilterBeta) * filteredGyroZ;
```

Then we integrate the gyroscope data to get the roll, pitch and yaw. To perform integration in C++, we will just simply multiply dt (=0.01f).

```cpp
// Integrate gyroscope data to get roll, pitch, and yaw
float gyro_roll = roll + filteredGyroX * dt;
float gyro_pitch = pitch + filteredGyroY * dt;
float gyro_yaw = yaw + filteredGyroZ * dt;
```

In order to find the pitch and roll from the IMU, the following equation will be used.

$$\text{pitch} = \text{atan2}(-\text{filteredAccX}, \sqrt{\text{filteredAccY}^2 + \text{filteredAccZ}^2})$$

$$\text{roll} = \text{atan2}(\text{filteredAccY}, \sqrt{\text{filteredAccX}^2 + \text{filteredAccZ}^2})$$

Code implementation for the mathematic equations to C++ language.

```cpp
// Calculate roll and pitch from accelerometer data
float acc_roll = atan2(filteredAccY, sqrt(filteredAccX * filteredAccX + filteredAccZ * filteredAccZ));
float acc_pitch = atan2(-filteredAccX, sqrt(filteredAccY * filteredAccY + filteredAccZ * filteredAccZ));
```

After printing the data in the serial port, you can see the difference between applying the filter and without the filter.