IE6-BUL S23

Function and Characterization of an Inertial Sensor Cluster/ I2C bus

Authors: Soodeh Mousaviasl, Celestine Machuca.

Materials used

- D1 Mini microcontroller
- Breadboard for prototyping
- MPU-6050 inertial sensor

Connection diagram

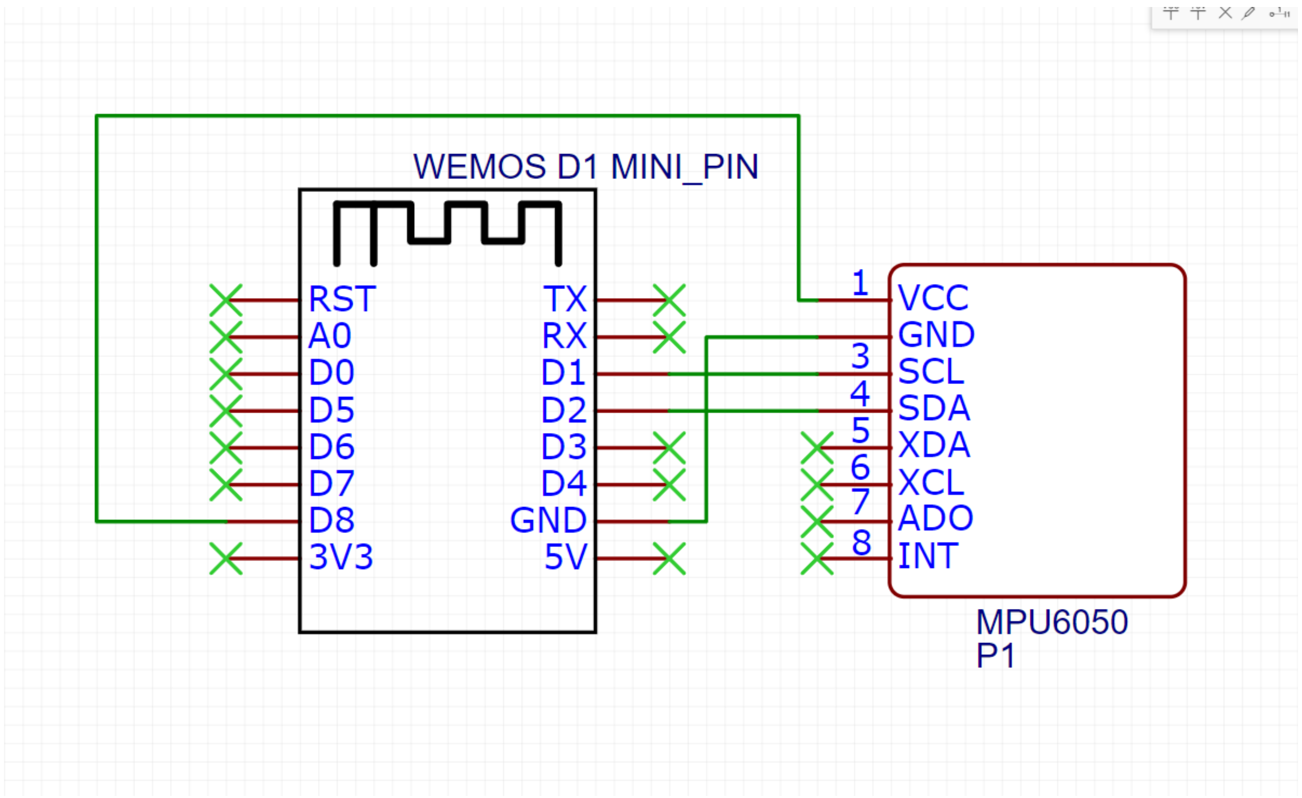


Fig 1 MPU-6050 connections

Setup Used

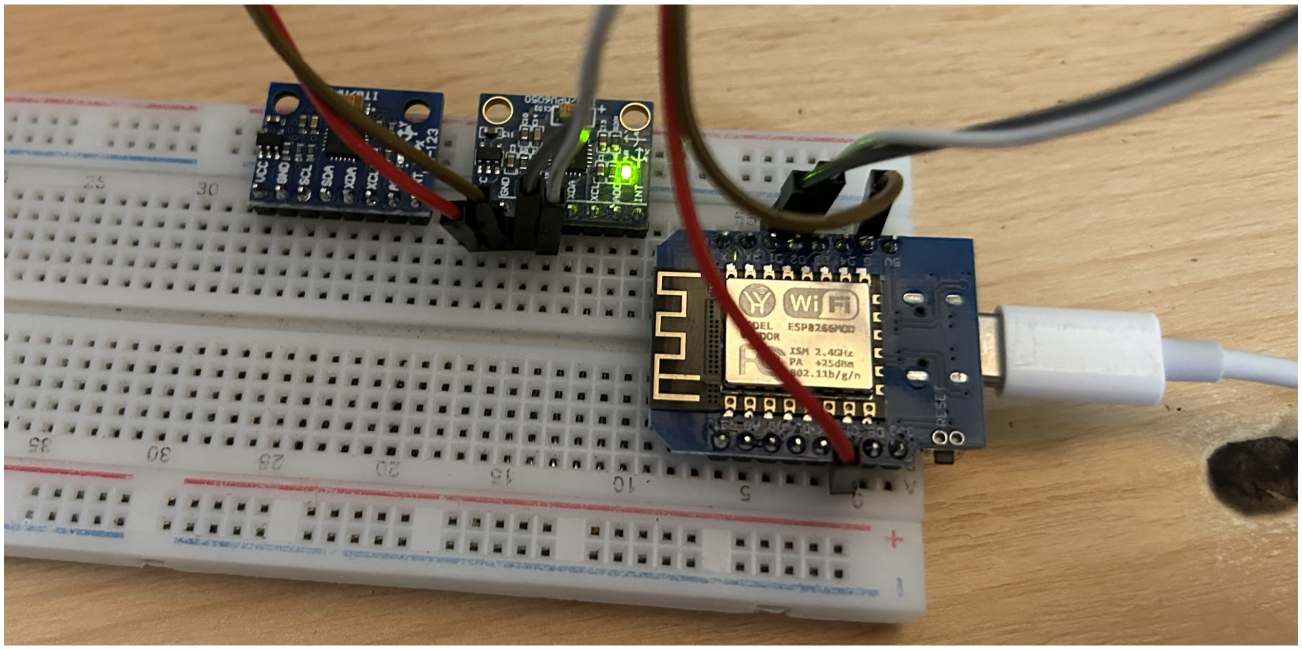


Fig 2 Setup

Lab Tasks

Part 1 Setup

- Which measurement data is recorded

c code used

```
#include <Arduino.h>
#include <Wire.h>
#define sensor_address 0x68

// keywords = ['@filter_setting', '@accelerometer_setting', '@gyroscope_setting']
#define FILTER_CONFIG_REG 0x1A

#define SET_FILTER_260HZ 0x06
#define SET_FILTER_184HZ 0x01
#define SET_FILTER_94HZ 0x02
#define SET_FILTER_44HZ 0x03
#define SET_FILTER_21HZ 0x04
#define SET_FILTER_10HZ 0x05
#define SET_FILTER_5HZ 0x06

#define GYRO_CONFIG_REG 0x1B

#define SET_GYRO_250 0x00
#define SET_GYRO_500 0x08
#define SET_GYRO_1000 0x10
#define SET_GYRO_2000 0x18

#define ACC_CONFIG_REG 0x1C

#define SET_ACC_2G 0x00
#define SET_ACC_4G 0x08
#define SET_ACC_8G 0x10
#define SET_ACC_16G 0x18

void SetConfiguration(byte reg, byte setting)
{
  // Aufruf des MPU6050 Sensor
  Wire.beginTransmission(sensor_address);
  // Register Aufruf
  Wire.write(reg);
  // Einstellungsbyte für das Register senden
  Wire.write(setting);
  Wire.endTransmission();
}

void setup()
{
  Serial.begin(115200);
  pinMode(D7, OUTPUT);
  digitalWrite(D7, LOW);
  delay(1000);
  digitalWrite(D7, HIGH);
  delay(1000);
  Wire.begin();
  delay(1000);
  // Powermanagement aufrufen
  // Sensor schlafen und Reset, Clock wird zunächst von Gyro-Achse Z verwendet
  // Serial.println("Powermanagement aufrufen - Reset");
  SetConfiguration(0x6B, 0x80);

  // Kurz warten
  delay(500);

  // Powermanagement aufrufen
  // Sleep beenden und Clock von Gyroskopeachse X verwenden
  // Serial.println("Powermanagement aufrufen - Clock festlegen");
  SetConfiguration(0x6B, 0x03);

  delay(500);
  // filter configuration
  SetConfiguration(FILTER_CONFIG_REG, @filter_setting);
  // gyro config
  SetConfiguration(GYRO_CONFIG_REG, @gyroscope_setting);
  // acc config
  SetConfiguration(ACC_CONFIG_REG, @accelerometer_setting);
  delay(500);
}

void loop()
{
  byte result[14];
  result[0] = 0x3B;
  Wire.beginTransmission(sensor_address);
  Wire.write(result[0]);
  Wire.endTransmission();
  Wire.requestFrom(sensor_address, 14);
  for (int i = 0; i < 14; i++)
  {
    result[i] = Wire.read();
  }

  // Accelerometer
  int16_t acc_X = (((int16_t)result[0]) << 8) | result[1];
  int16_t acc_Y = (((int16_t)result[2]) << 8) | result[3];
  int16_t acc_Z = (((int16_t)result[4]) << 8) | result[5];

  // Temperature sensor
  int16_t temp = (((int16_t)result[6]) << 8) | result[7];
  int16_t tempC = temp / 340 + 36.53;

  // Gyroscope
  int16_t gyr_X = (((int16_t)result[8]) << 8) | result[9];
  int16_t gyr_Y = (((int16_t)result[10]) << 8) | result[11];
  int16_t gyr_Z = (((int16_t)result[12]) << 8) | result[13];
  // Print data
```

```
    // json like format
    Serial.print("{\"acc_X\":");
    Serial.print(acc_X);
    Serial.print(",\"acc_Y\":");
    Serial.print(acc_Y);
    Serial.print(",\"acc_Z\":");
    Serial.print(acc_Z);
    Serial.print(",\"temp\":");
    Serial.print(temp);
    Serial.print(",\"tempC\":");
    Serial.print(tempC);
    Serial.print(",\"gyr_X\":");
    Serial.print(gyr_X);
    Serial.print(",\"gyr_Y\":");
    Serial.print(gyr_Y);
    Serial.print(",\"gyr_Z\":");
    Serial.print(gyr_Z);
    Serial.println("}");
  }
```

From the code the data being recorded is:

```
// Accelerometer
int16_t acc_X = (((int16_t)result[0]) << 8) | result[1];
int16_t acc_Y = (((int16_t)result[2]) << 8) | result[3];
int16_t acc_Z = (((int16_t)result[4]) << 8) | result[5];

// Temperature sensor
int16_t temp = (((int16_t)result[6]) << 8) | result[7];
int16_t tempC = temp / 340 + 36.53;

// Gyroscope
int16_t gyr_X = (((int16_t)result[8]) << 8) | result[9];
int16_t gyr_Y = (((int16_t)result[10]) << 8) | result[11];
int16_t gyr_Z = (((int16_t)result[12]) << 8) | result[13];
```

Example output:

```
{"acc_X":4210,"acc_Y":15,"acc_Z":-1416,"temp":-4341,"tempC":24,"gyr_X":-83,"gyr_Y":-34,"gyr_Z":36}
{"acc_X":4212,"acc_Y":-2,"acc_Z":-1401,"temp":-4312,"tempC":24,"gyr_X":-85,"gyr_Y":-34,"gyr_Z":38}
{"acc_X":4214,"acc_Y":-4,"acc_Z":-1416,"temp":-4335,"tempC":24,"gyr_X":-85,"gyr_Y":-37,"gyr_Z":34}
{"acc_X":4207,"acc_Y":7,"acc_Z":-1409,"temp":-4332,"tempC":24,"gyr_X":-88,"gyr_Y":-37,"gyr_Z":33}
{"acc_X":4194,"acc_Y":1,"acc_Z":-1385,"temp":-4331,"tempC":24,"gyr_X":-84,"gyr_Y":-36,"gyr_Z":34}
{"acc_X":4216,"acc_Y":13,"acc_Z":-1412,"temp":-4332,"tempC":24,"gyr_X":-87,"gyr_Y":-37,"gyr_Z":33}
{"acc_X":4208,"acc_Y":15,"acc_Z":-1405,"temp":-4330,"tempC":24,"gyr_X":-90,"gyr_Y":-37,"gyr_Z":35}
{"acc_X":4209,"acc_Y":7,"acc_Z":-1389,"temp":-4343,"tempC":24,"gyr_X":-91,"gyr_Y":-40,"gyr_Z":34}
{"acc_X":4211,"acc_Y":10,"acc_Z":-1393,"temp":-4332,"tempC":24,"gyr_X":-88,"gyr_Y":-35,"gyr_Z":33}
{"acc_X":4194,"acc_Y":-1,"acc_Z":-1399,"temp":-4331,"tempC":24,"gyr_X":-86,"gyr_Y":-38,"gyr_Z":35}
```

- how does the measurement data get to the laptop?

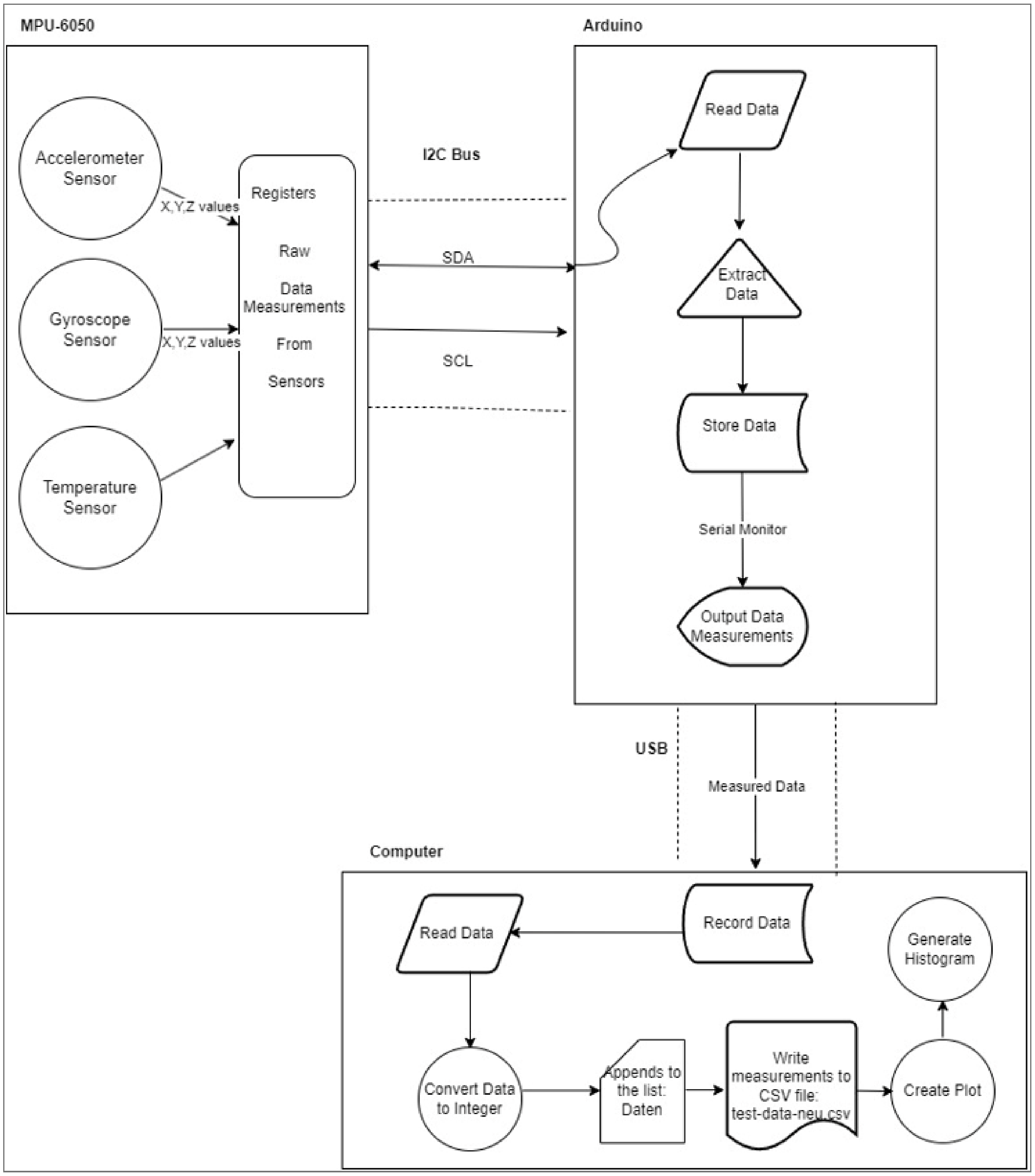


Fig 3 Flow Chart

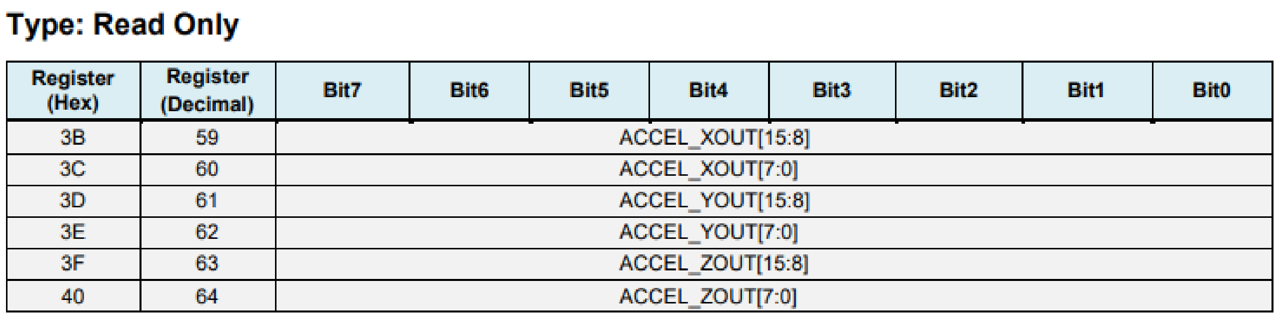
The MPU6050 sends data from its gyro, accelerometer, and temperature sensor to the Arduino via the I2C bus.
The Arduino processes the received data and formats it for transmission.
The Arduino sends the formatted data to the laptop via a USB cable using serial communication.
A serial terminal or custom software on the laptop reads and interprets the received data, allowing you to view and analyze the measurements.

Part 2 Configuration

- Find out how to set the sensor cluster to different bandwidths and measurement ranges by analyzing the data sheet?

Using the datasheet, the sensor cluster can be set to different bandwidth and measurements through modifying the values of specific registers in the MPU-6050's register map.
The 0x1A register – CONFIG – is used to configure the DLPF (Digital Low-Pass Filter) and the sampling rate divider (SMPLRT_DIV). The 0x1B register - GYRO_CONFIG – is for configuring the Gyroscope's full-scale range. And the 0x1C register - ACCEL_CONFIG – is for configuring the full-scale range of the accelerometer.
Hence, to set different bandwidth the DLPF_CFG bits of 0x1A register were modified based on the availble options (look datasheet register map page 13).
On Page 29 to 31 of the MPU6050 register map, there is a table that shows the different configurations for the accelerometer and gyroscope. The table shows the different bandwidths and measurement range and the corresponding register values.

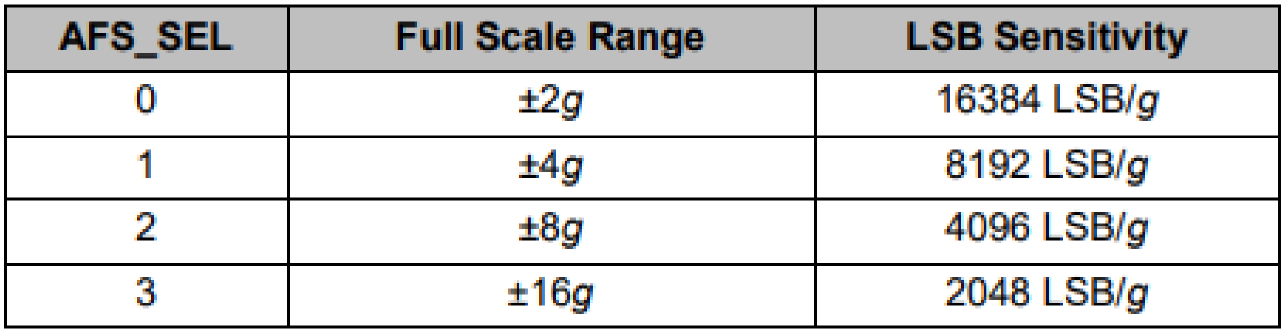
Page 29 Registers 59 to 64 – Accelerometer Measurements



Type: Read Only

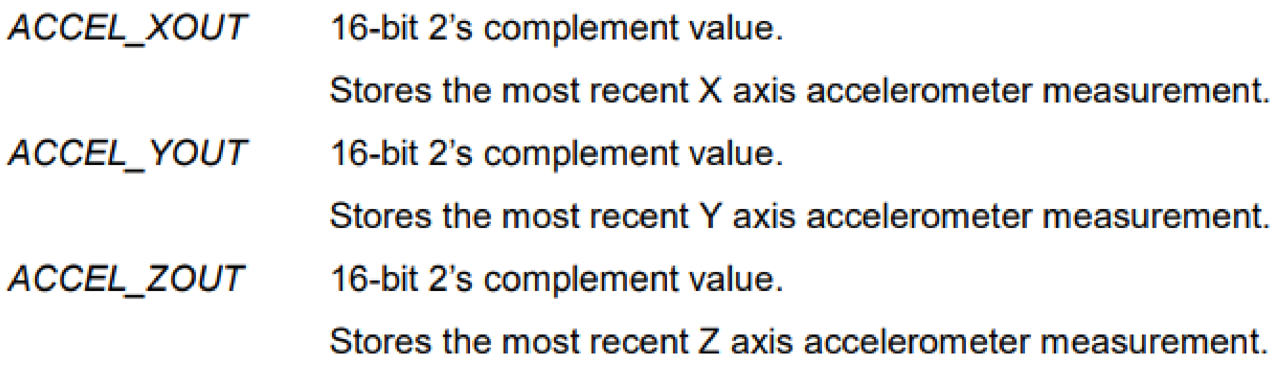
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

Fig 4 Accelerometer



AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

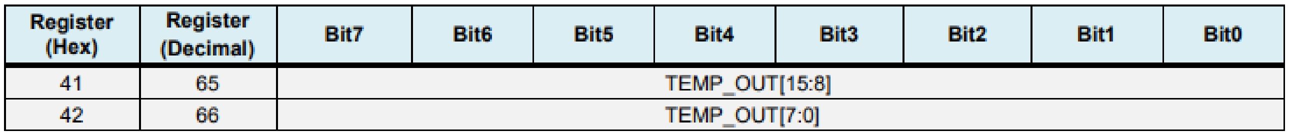
Fig 5 Accelerometer Bandwidth



ACCEL_XOUT	16-bit 2's complement value.
	Stores the most recent X axis accelerometer measurement.
ACCEL_YOUT	16-bit 2's complement value.
	Stores the most recent Y axis accelerometer measurement.
ACCEL_ZOUT	16-bit 2's complement value.
	Stores the most recent Z axis accelerometer measurement.

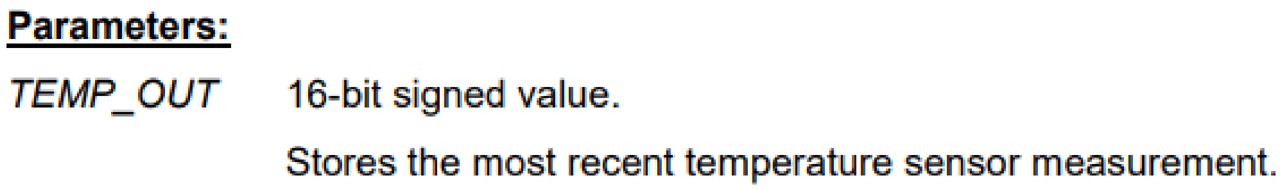
Fig 6 Accelerometer Parameters

Page 29 Registers 65 to 66 – Temperature Measurements



Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
41	65	TEMP_OUT[15:8]							
42	66	TEMP_OUT[7:0]							

Fig 7 Temperature



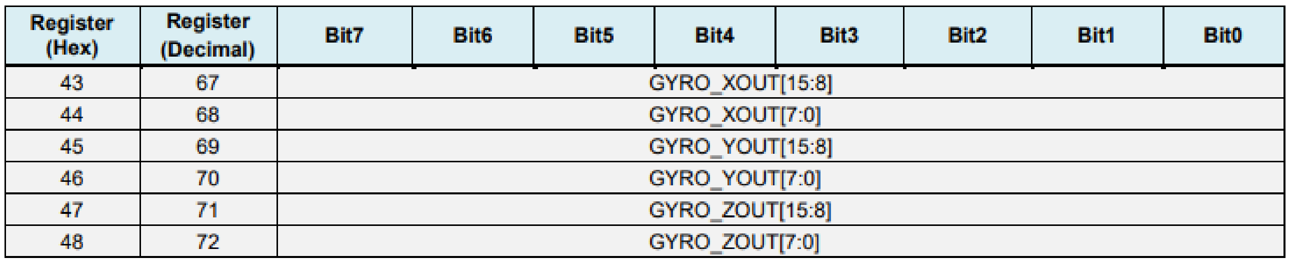
Parameters:

TEMP_OUT	16-bit signed value.
	Stores the most recent temperature sensor measurement.

Fig 8 Temperature Parameters

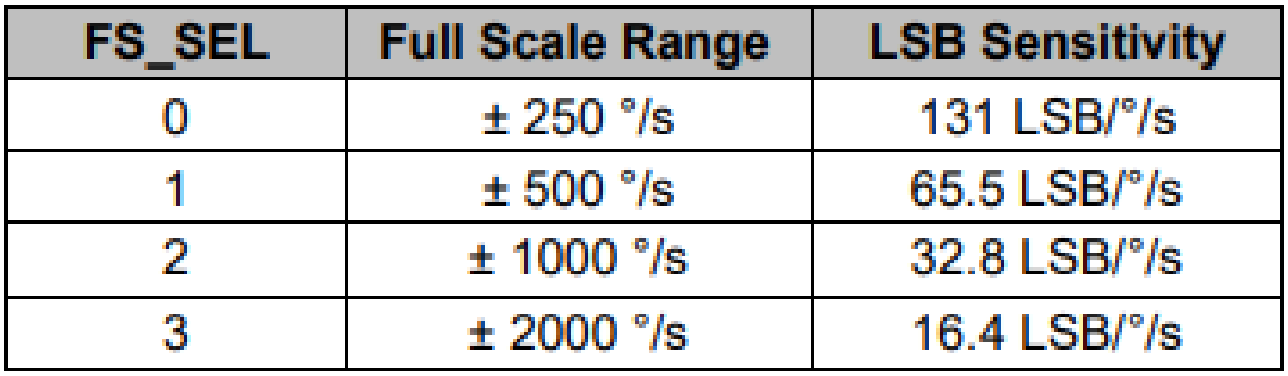
Using the datasheet, the sensor cluster can be set to different bandwidth and measurements through modifying the values of specific registers in the MPU-6050's register map.
The 0x1A register – CONFIG – is used to configure the DLPF (Digital Low-Pass Filter) and the sampling rate divider (SMPLRT_DIV). The 0x1B register - GYRO_CONFIG – is for configuring the Gyroscope's full-scale range. And the 0x1C register - ACCEL_CONFIG – is for configuring the full-scale range of the accelerometer.
Hence, to set different bandwidth the DLPF_CFG bits of 0x1A register were modified based on the availble options (look datasheet register map page 13).

Page 30 Registers 67 to 72 – Gyroscope Measurements



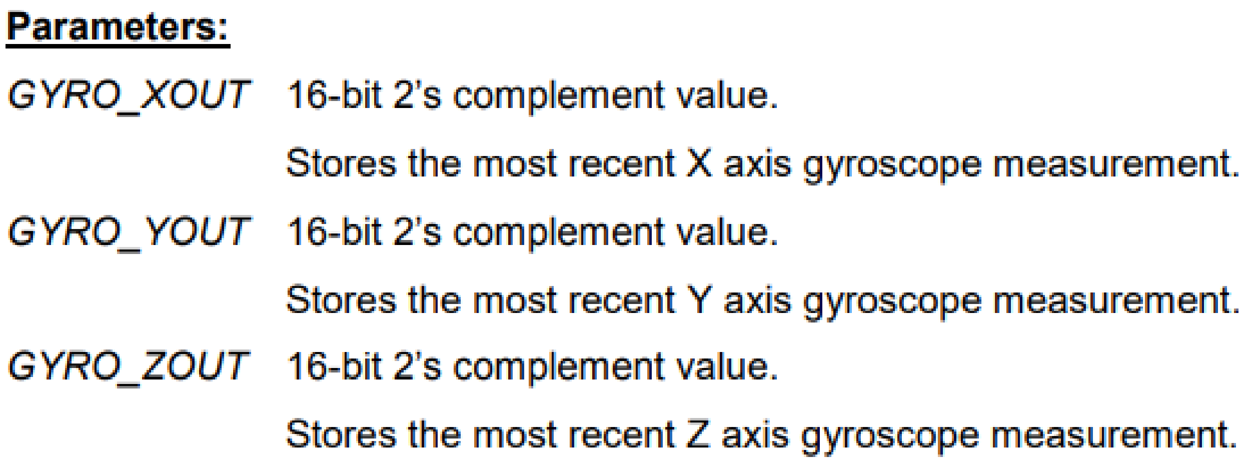
Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

Fig 7 Gyroscope



FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Fig 8 Gyroscope Bandwidth



Parameters:

GYRO_XOUT	16-bit 2's complement value. Stores the most recent X axis gyroscope measurement.
GYRO_YOUT	16-bit 2's complement value. Stores the most recent Y axis gyroscope measurement.
GYRO_ZOUT	16-bit 2's complement value. Stores the most recent Z axis gyroscope measurement.

Fig 9 Gyroscope Parameters

- Try out different configurations for the measuring range of one channel of the accelerometer and measure the digital output values for a = -1 g; 0 ; +1 g.

We performed the test over this values = +-2g, +-4g,+-8g, +-16h against earth gravity on the X Accelerometer. The results are shown below.

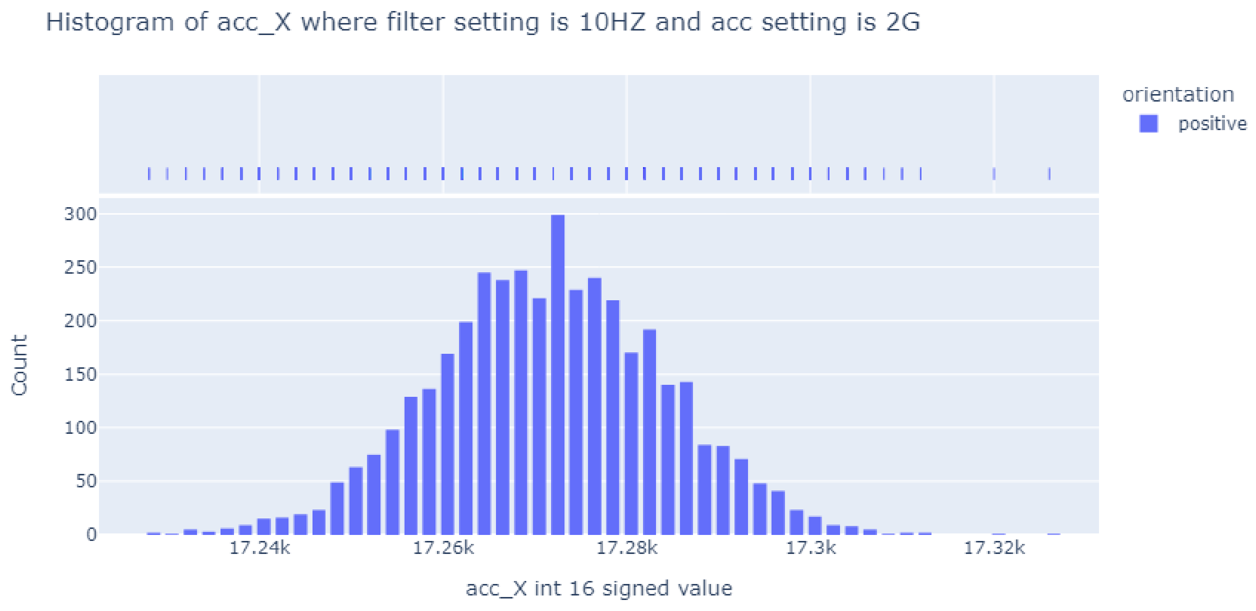


Fig 10 Histogram 10hz 2g against earth gravity

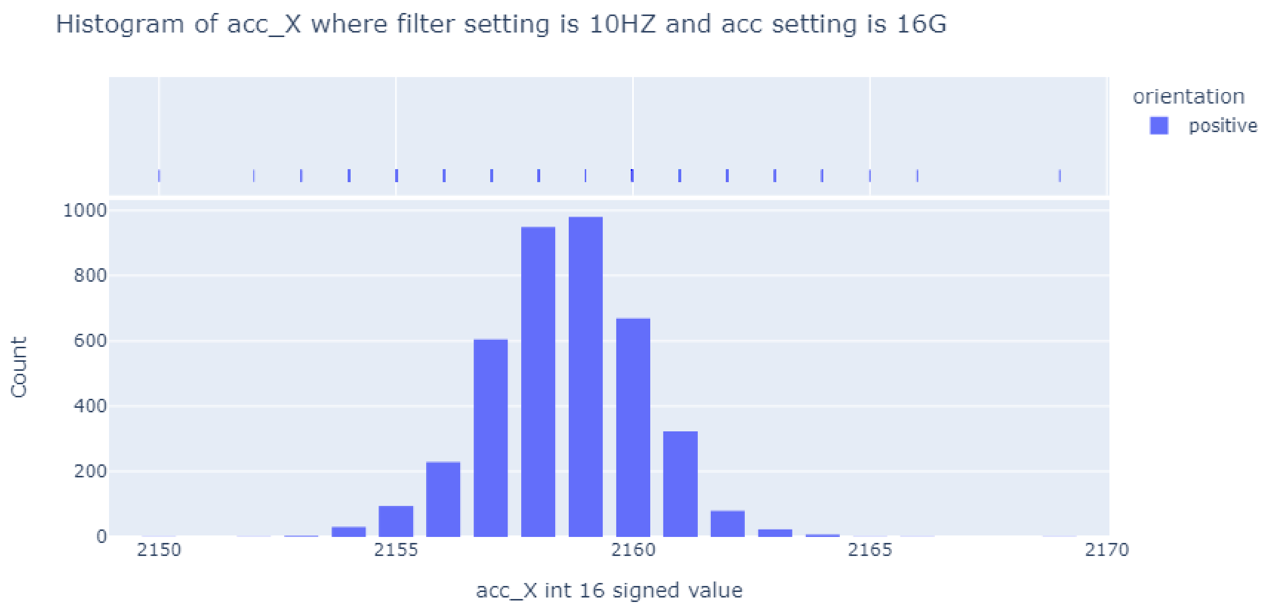


Fig 11 Histogram 10hz 16g against earth gravity

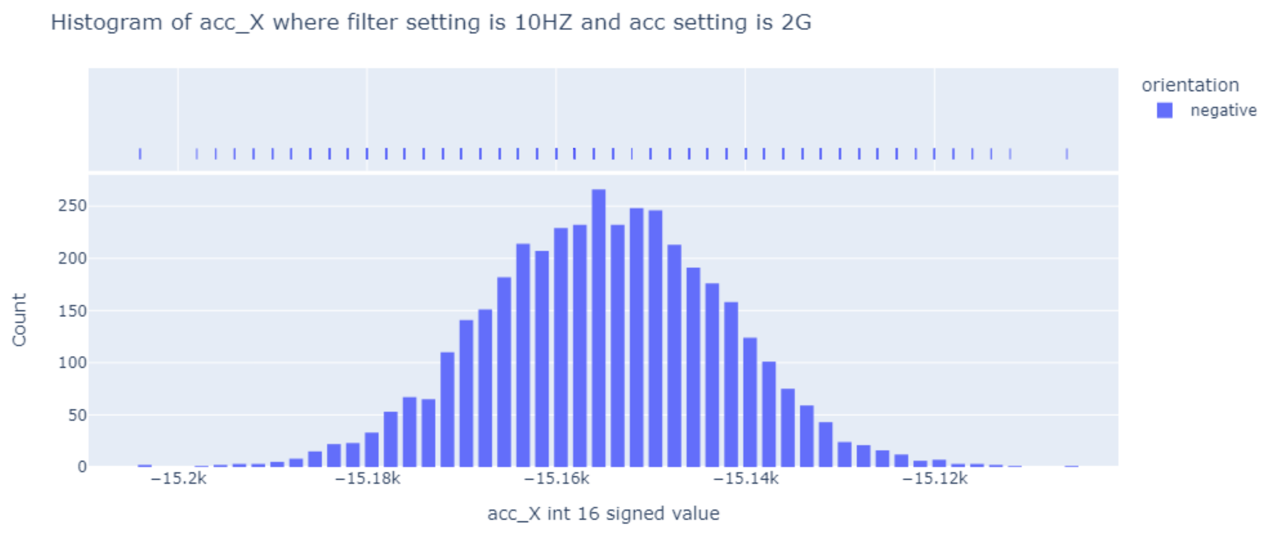


Fig 12 Histogram 10hz 2g with earth gravity

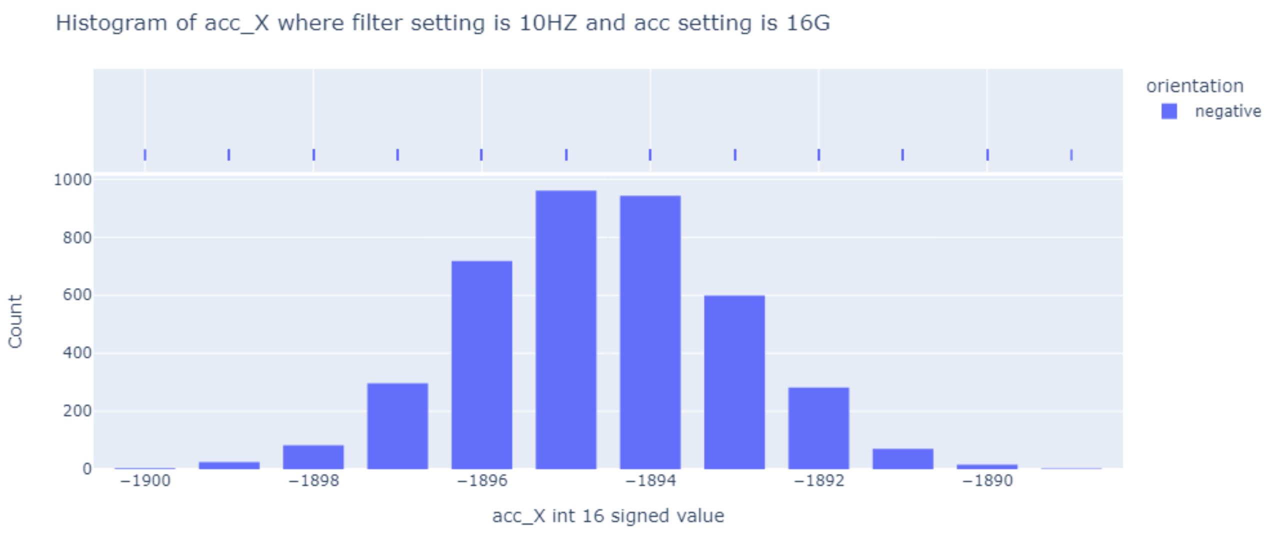


Fig 13 Histogram 10hz 16g with earth gravity

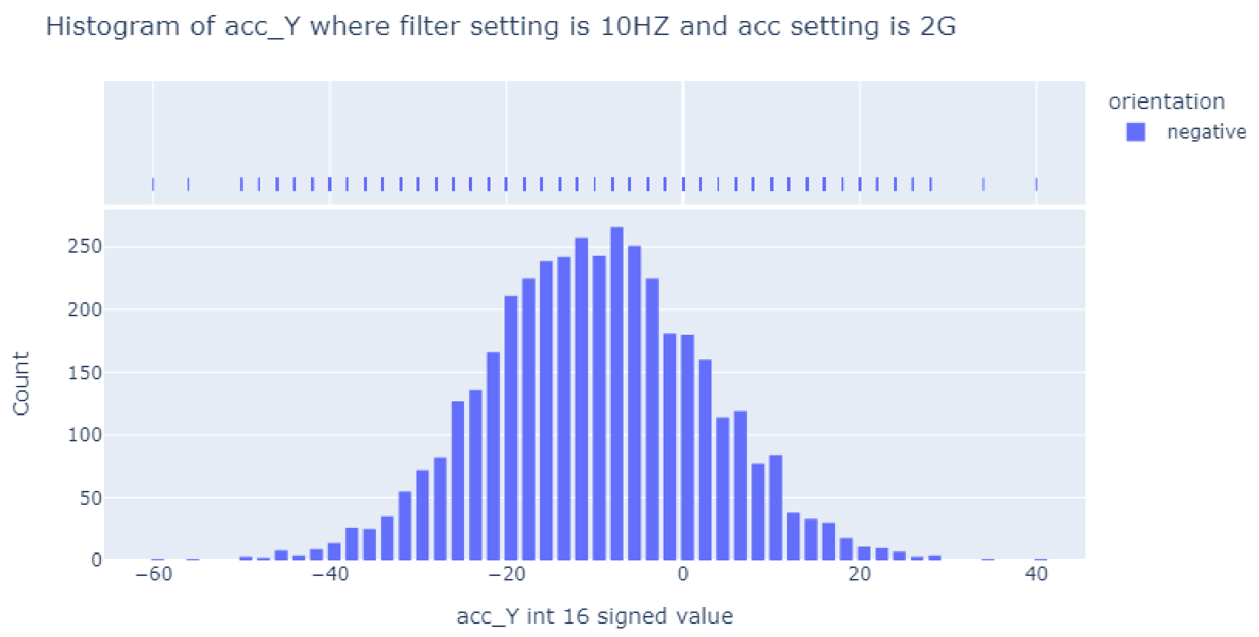


Fig 14 Histogram 10hz 2g neutral

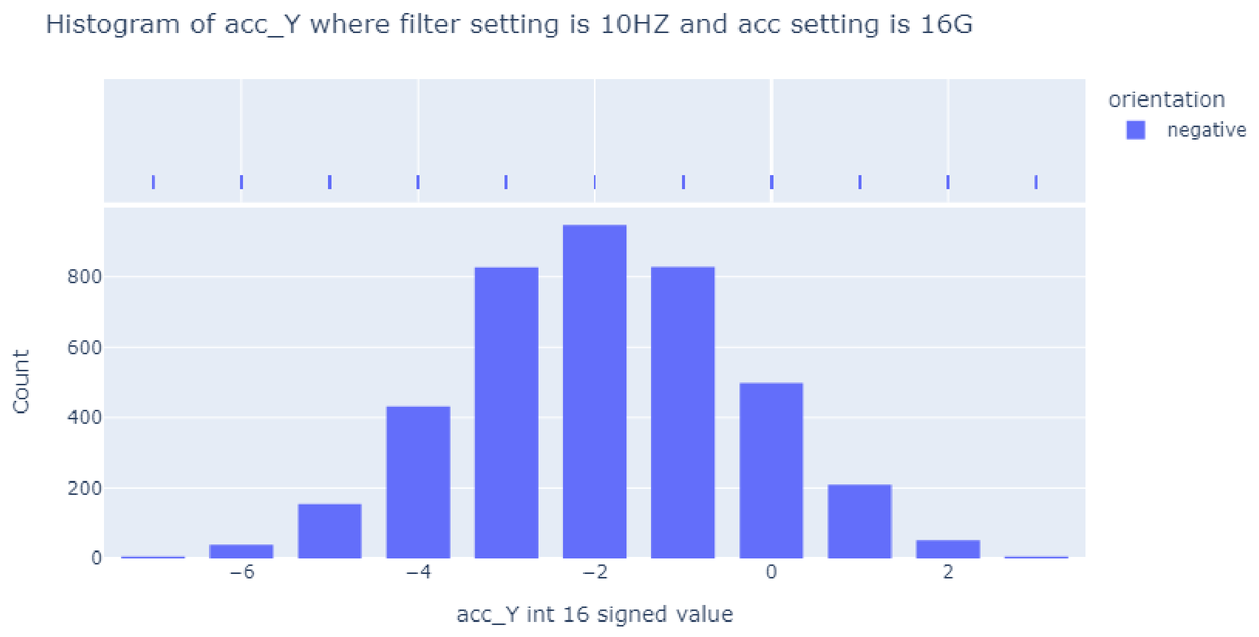


Fig 15 Histogram 10hz 16g neutral

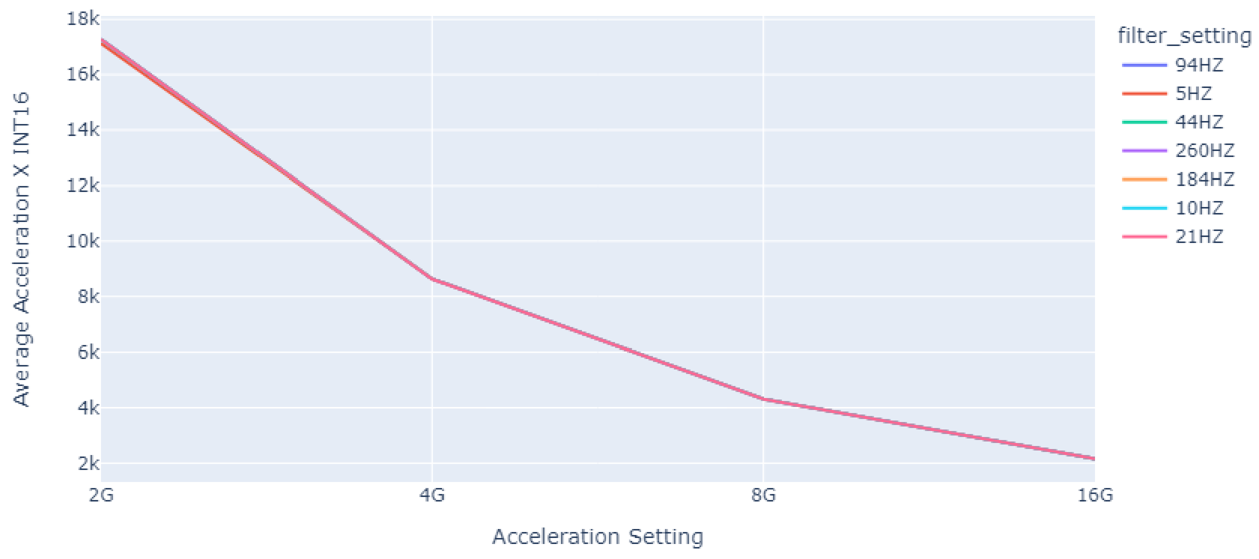


Fig 10 Accelerometer Results

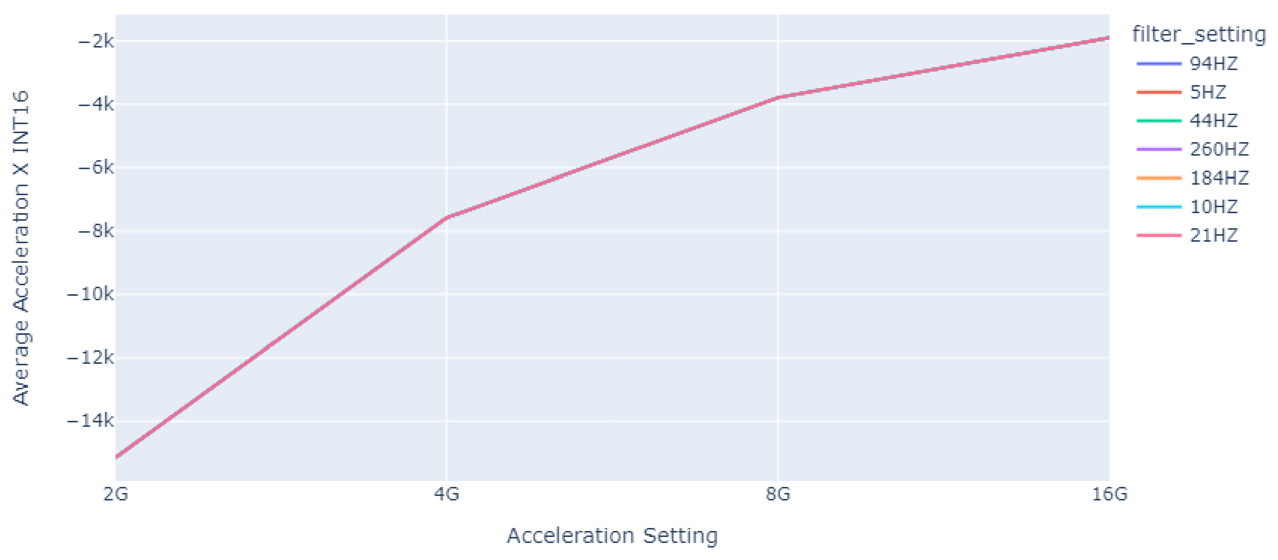


Fig 11 Accelerometer Results Negative

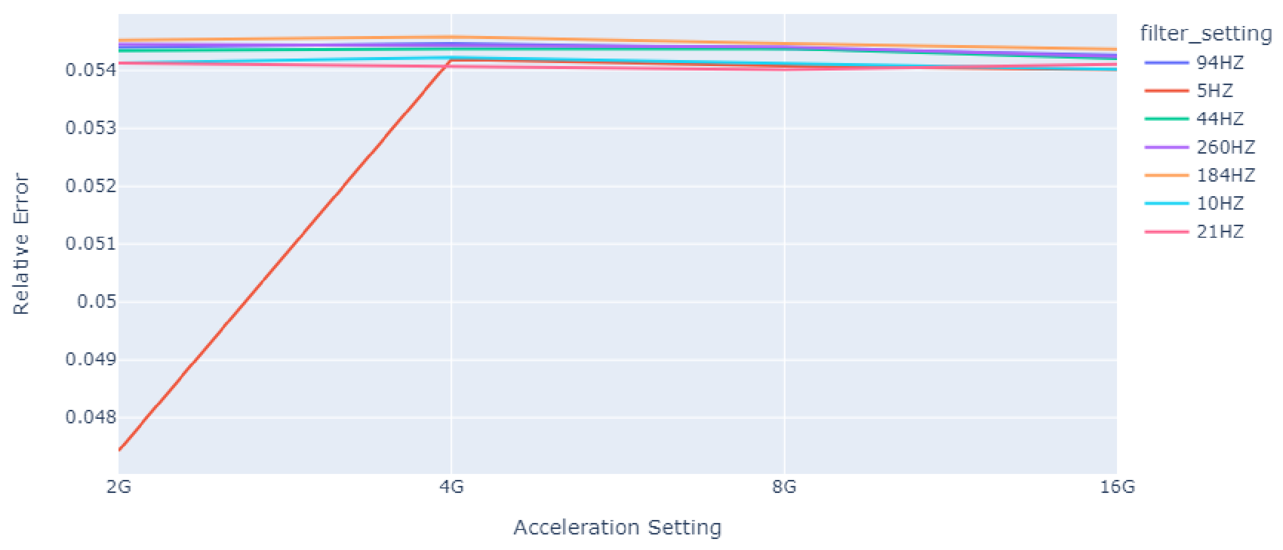


Fig 11 Accelerometer Relative Error

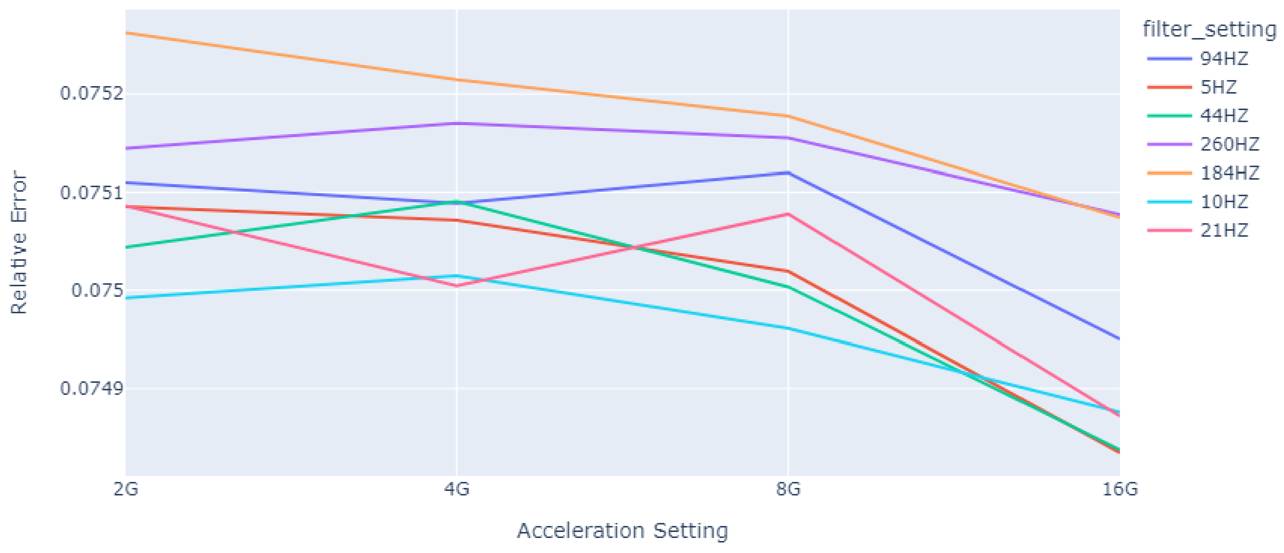


Fig 12 Accelerometer Relative Error Negative

- What is the resolution of each of these measurements?

given the range int 16 signed from -32768 to 32767, the resolution is 2^16/2 = 32768/2 = 16384 for 1g. 1g/16384 = 6.10E-05 g/LSB

changing the range translates on to the table below against the theoretical resolution and the measured resolution.

acc_setting	theoretical_resolution[g/LSB]	measured_resolution[g/LSB]
2G	6.10E-05	5.79E-05
4G	1.22E-04	1.16E-04
8G	2.44E-04	2.32E-04
16G	4.88E-04	4.63E-04

Part 3 Oscilloscope measurements on the I2C bus

- Connect the oscilloscope to your measurement setup. Measure the voltage between SCL and GND with a probe (please adjust the square-wave signal first!) and examine the data line SDA with a second probe.
- Compare your measurement result with the I2C data protocol from the lecture (or e.g. from https://de.wikipedia.org/wiki/I²C).

- Please answer the following questions in your lab report:What is the datarate? How many bits (raw) are transferred per second?Analyze a single I2C telegram based on your oscilloscope measurement.How does your measurement compare tothe physical layer of the ideal I2C?

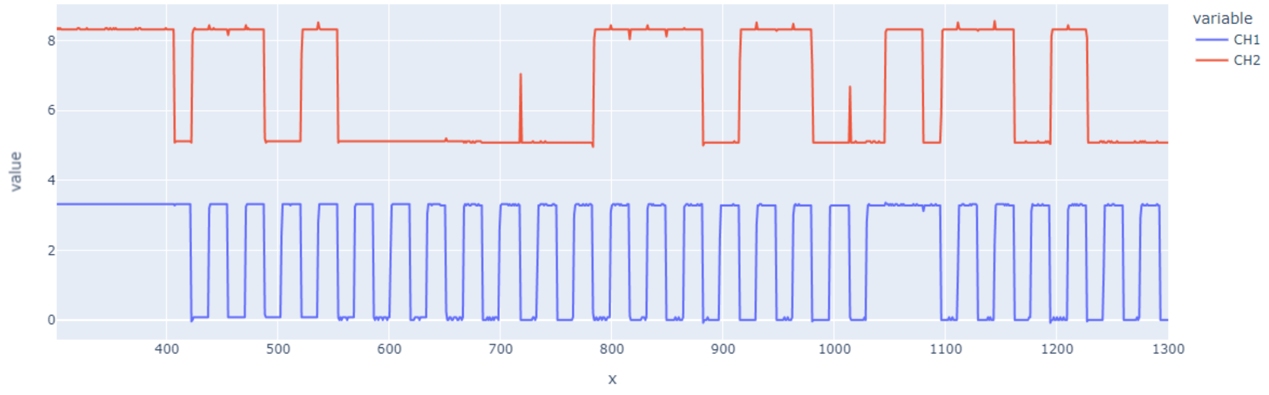


Fig 13 Oscilloscope

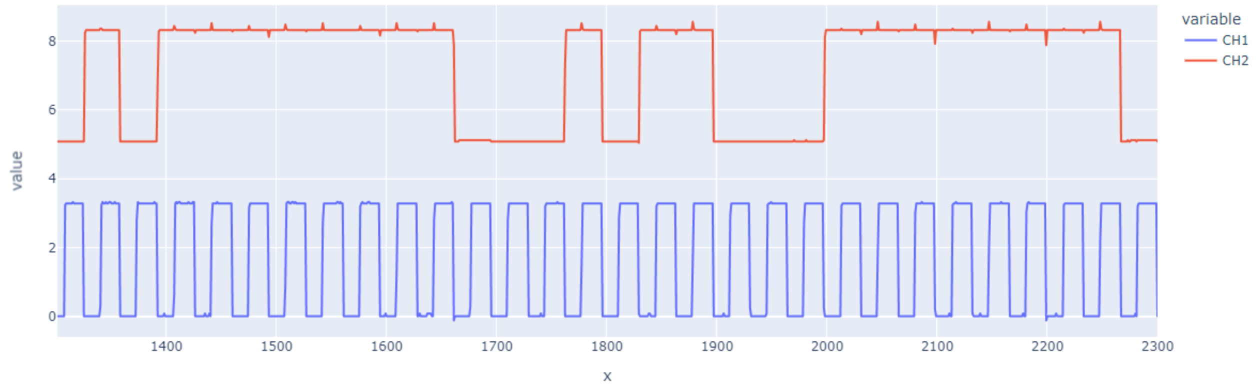


Fig 14 Oscilloscope

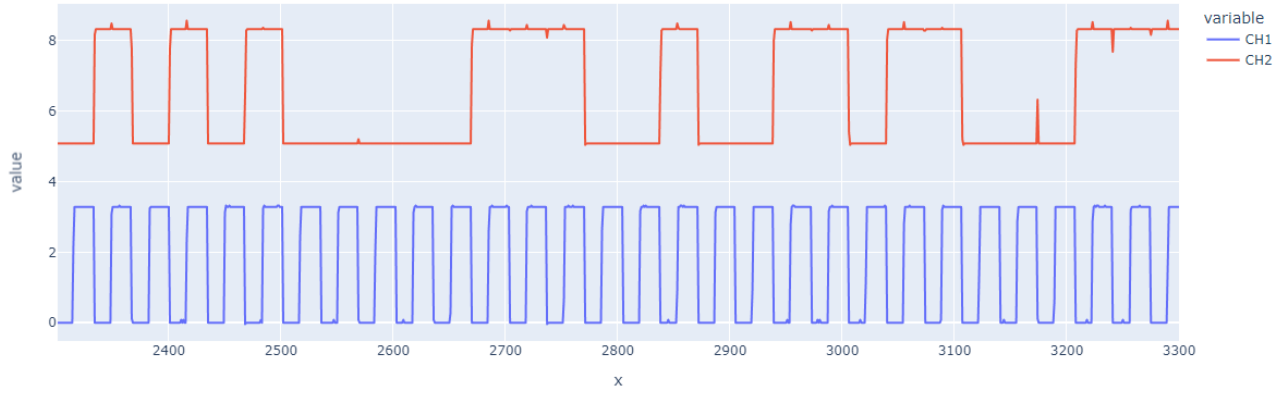


Fig 15 Oscilloscope

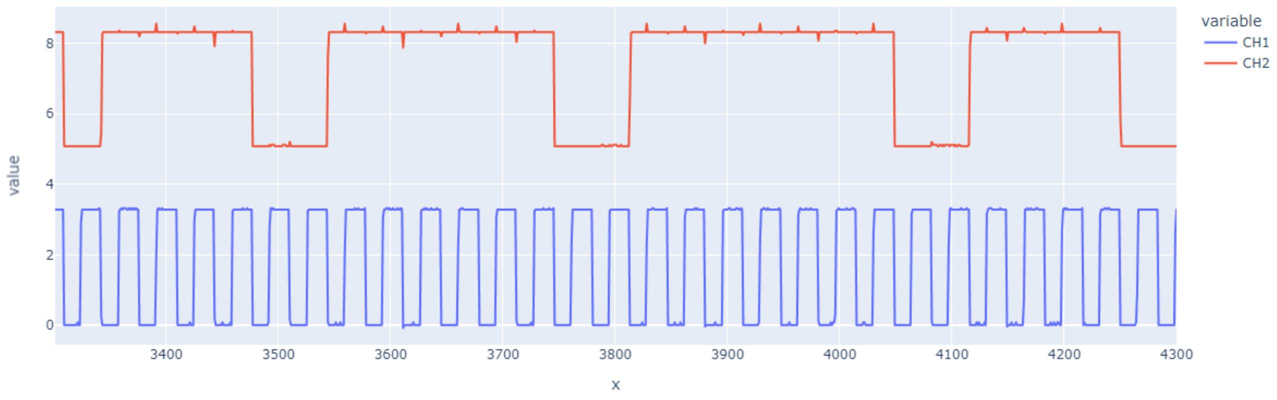


Fig 16 Oscilloscope

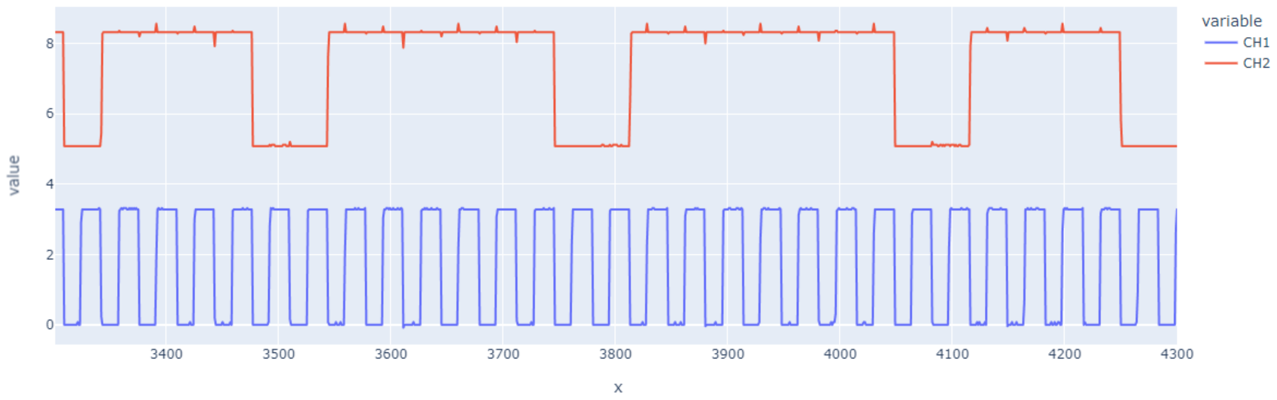


Fig 17 Oscilloscope

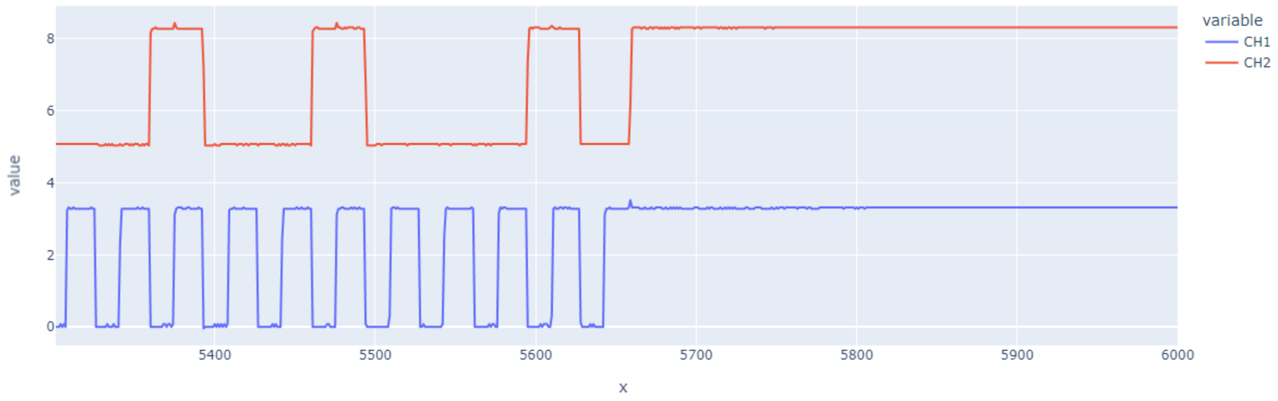


Fig 18 Oscilloscope

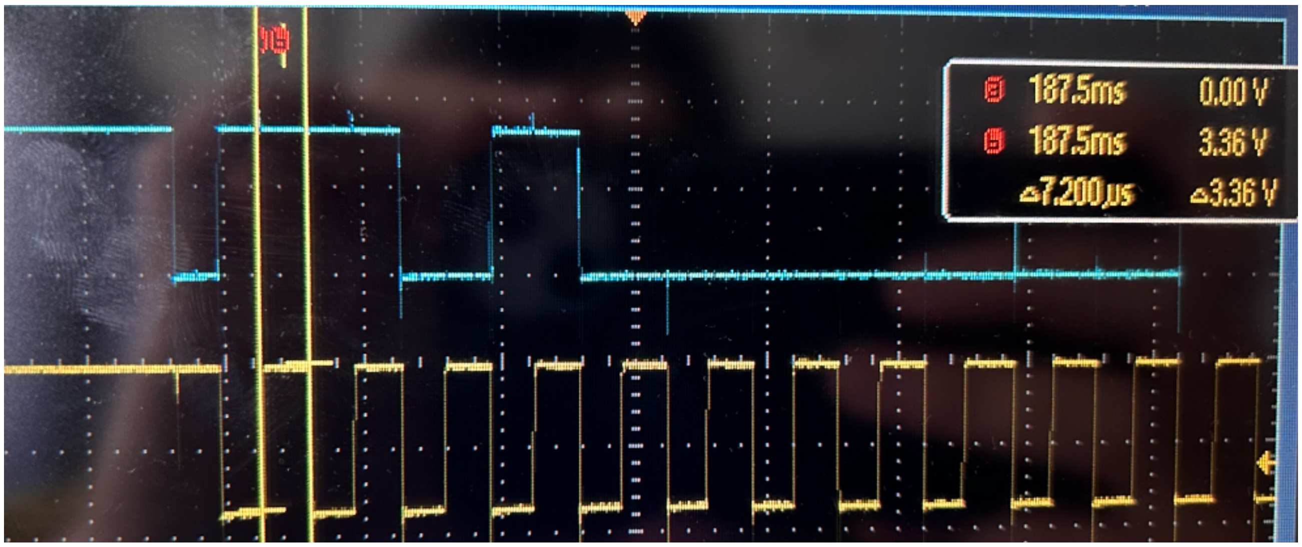


Fig 19 Bitrate

From the data collected from the oscilloscope, we can see that given the period of scl(CH1) of delta 7.2 microseconds * 2 = 14.4 microseconds, 1/14.4 * 10 ^ 6 = 69.44 kHz. This is the frequency of the clock signal, if we assumed 1 bit per clock cycle, then the bitrate would be 69.44 kbps but that would be without taking into account the overhead from the protocol and other factors.

```
The I2C protocl consists of following bits:
- start bit
- 7 bit slave (device) address
- Read/Write bit
- ACK bit
- 8 bit internal register address
- ACK bit
- 8 bit Data
- ACK bit
- Stop bit
source: https://www.youtube.com/watch?v=6IAkYpmA1DQ&t=68s
```

Part 4 Measuring Noise on a acceleration sensor

- Analyze the noise performance of one of the three axes of the accelerometer for different bandwidths (e.g. 260 Hz vs. 5 Hz)
- To do this, keep the sensor vibration-free/still and carry out a long-term measurement, e.g. over 1000 values. Check the measurement in the time domain for outliers, filter them out if necessary,using a suitable filter(either on the Arduino or on your computer).Document the filter and include source code and explanation into your report.
- Create and compare the histograms for at least two different bandwidths and, if applicable, with or w/o filter. What are the reasons for the differences?

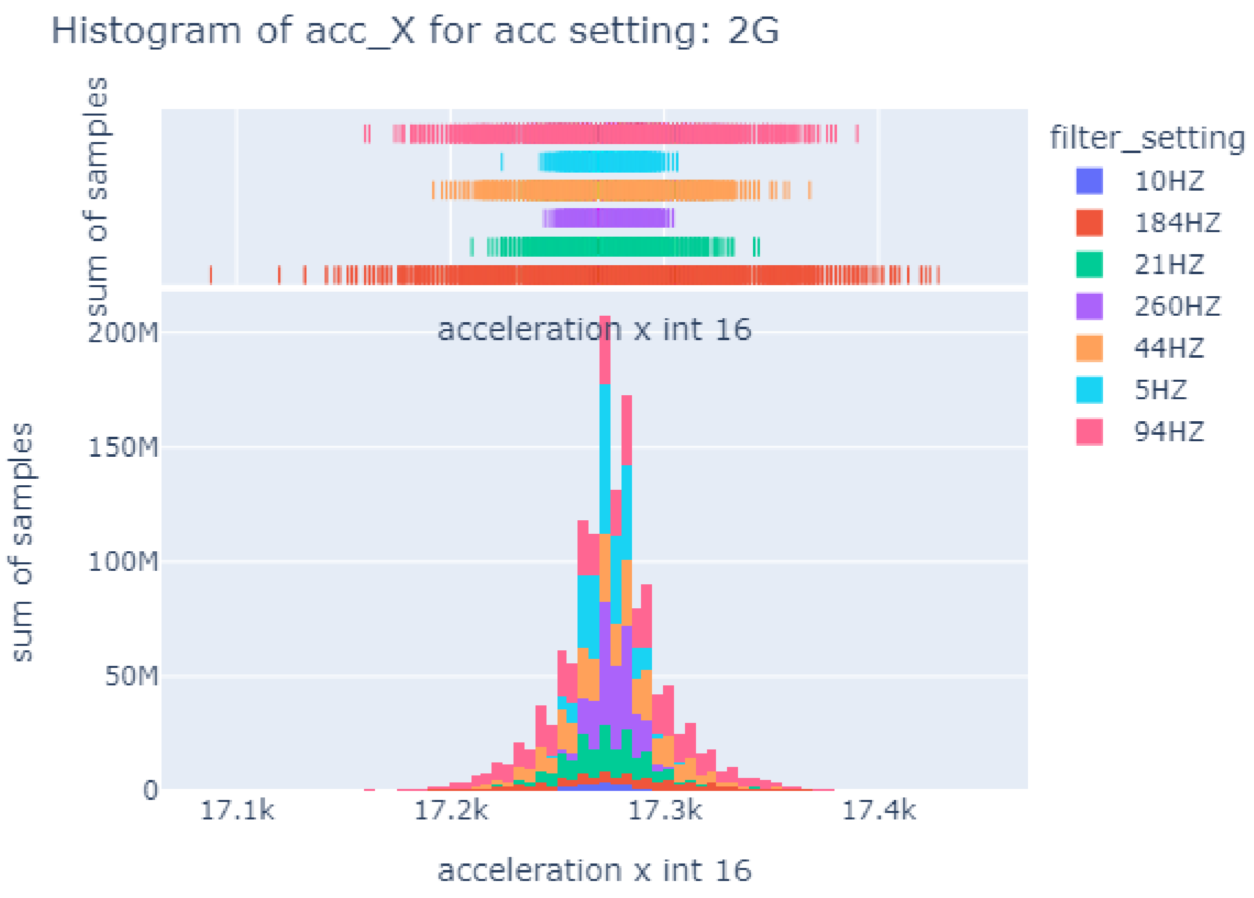


Fig 11 Histogram Acceleration X 2g Different Filters

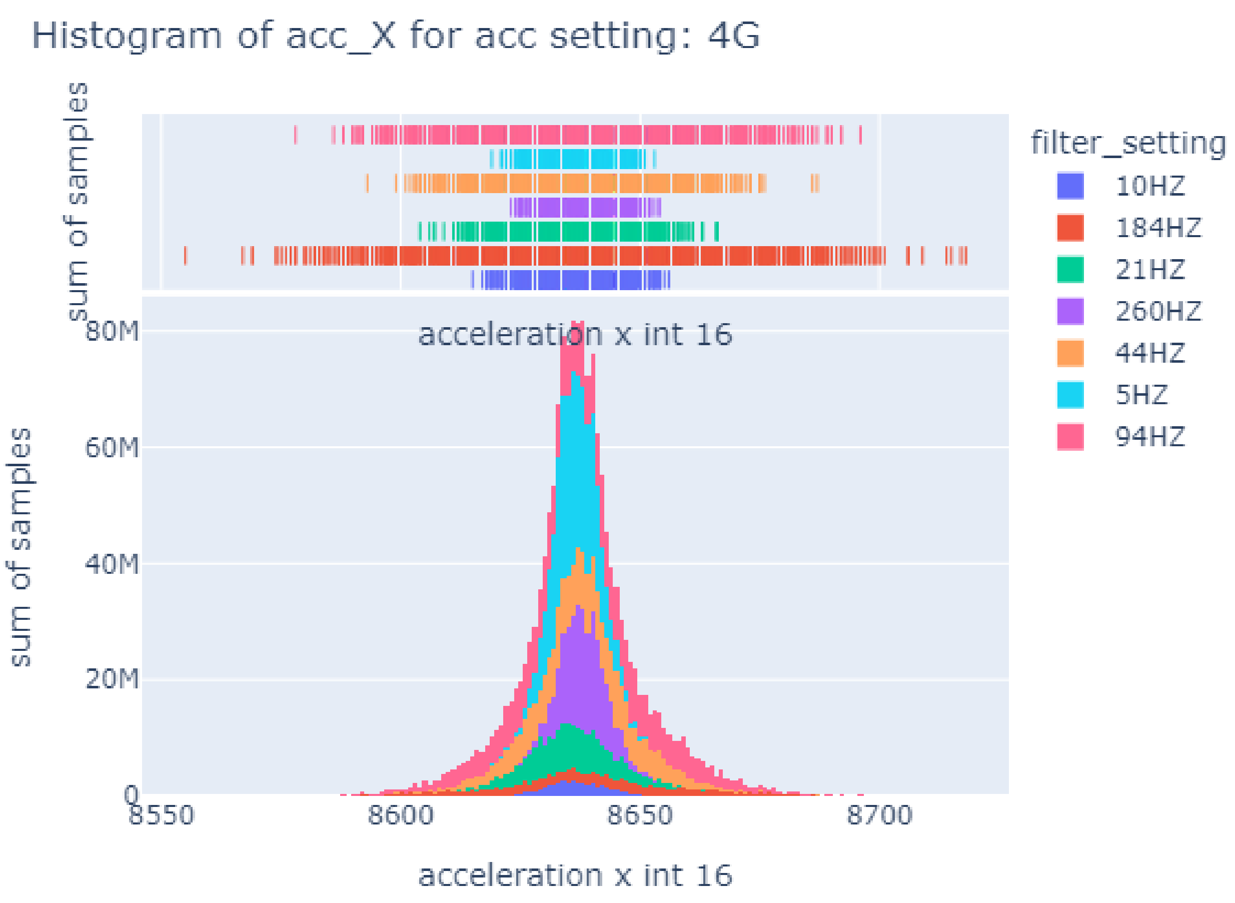


Fig 12 Histogram Acceleration X 4g Different Filters

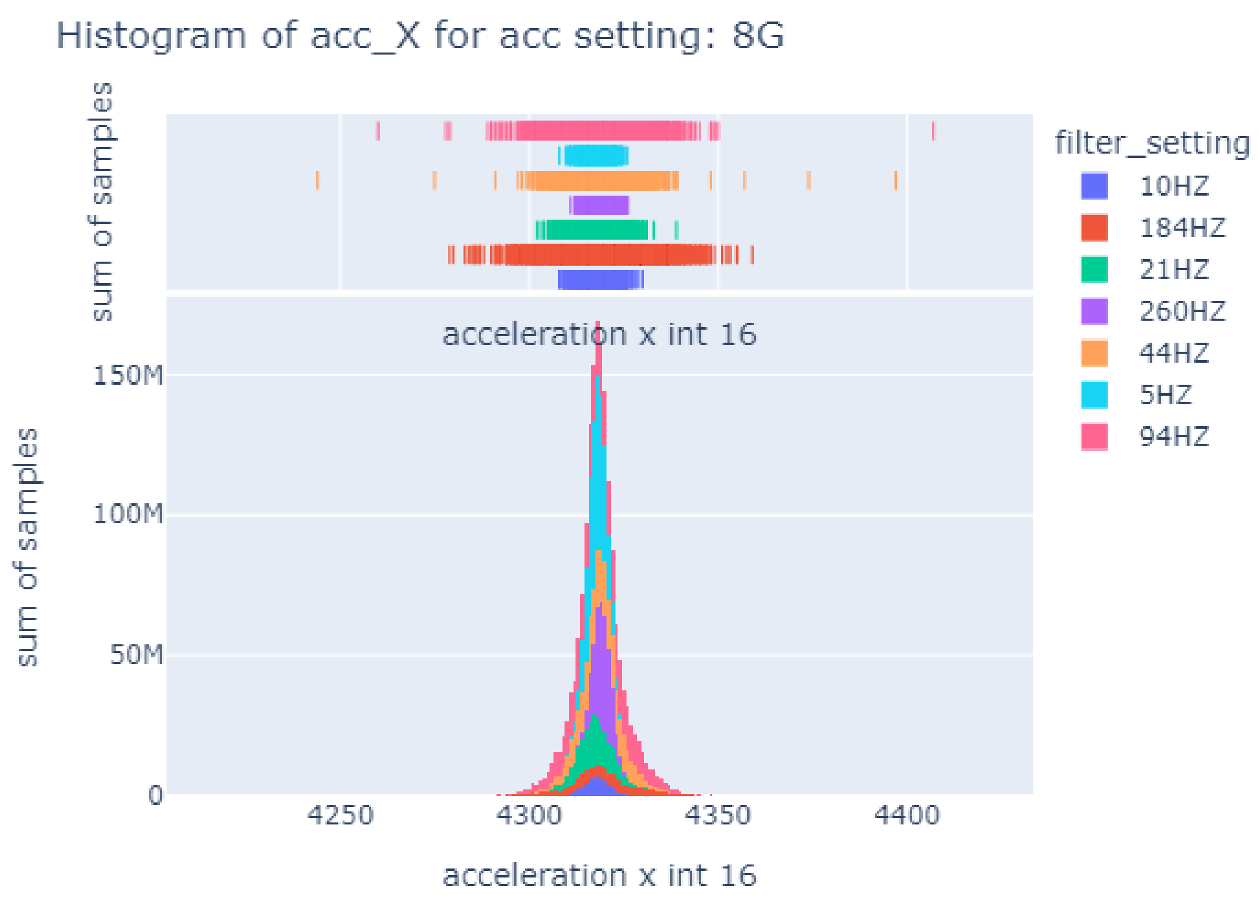


Fig 13 Histogram Acceleration X 8g Different Filters

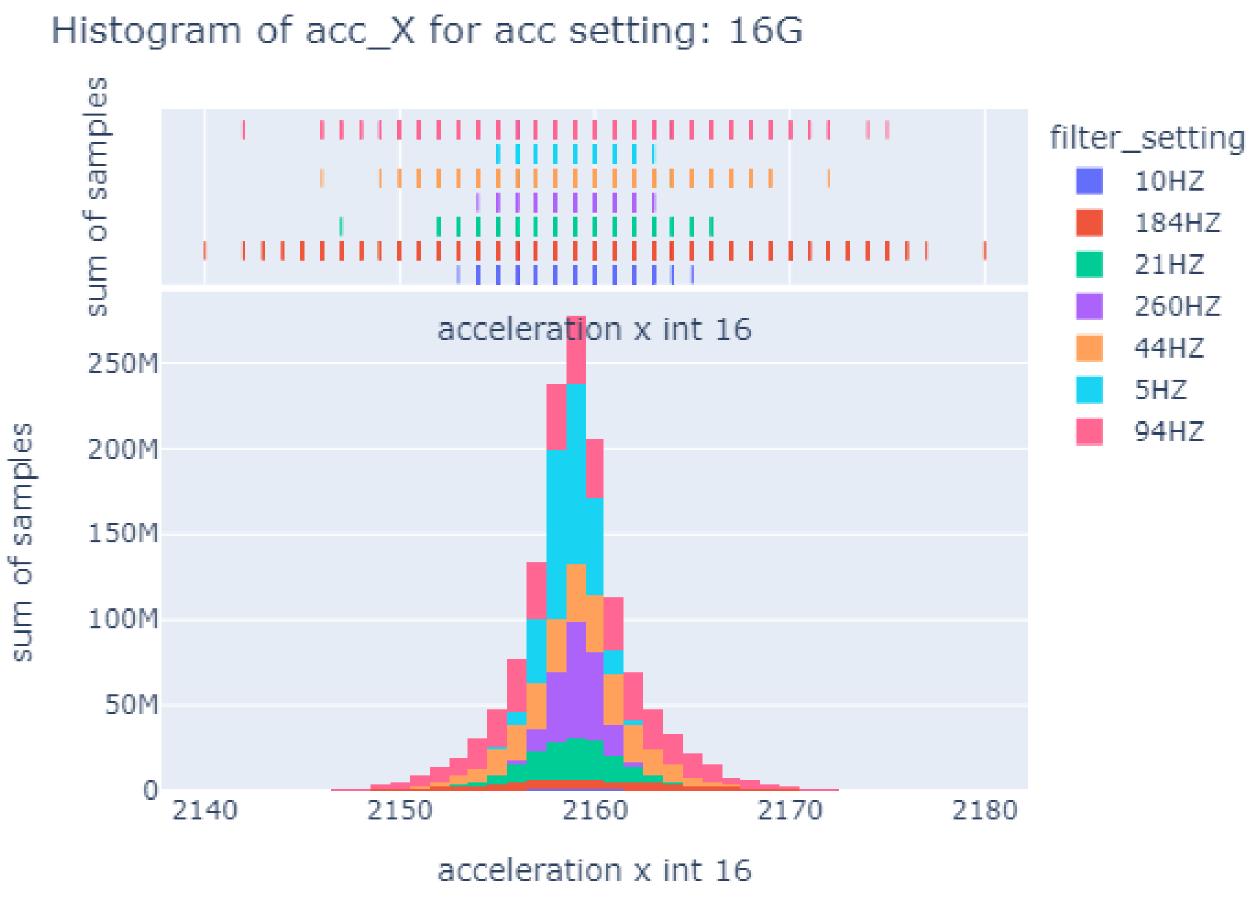


Fig 14 Histogram Acceleration X 16g Different Filters

There is no correlation in between the cutoff frequency on the low pass filter and the spread of the extreme values, as we originally expected.

Part 5 Determination of the noise behavior of a channel of the angular rate sensor -->

- Analyze the noise behavior of one of the three axes of the angular rate sensor for different bandwidths (e.g. 260 Hz vs. 5 Hz).
- To do this, keep the sensor vibration-free/still and carry out a long-term measurement, e.g. over 1000 values. Check the measurement in the time domain for outliers, filter them out if necessary,using a suitable filter in the Arduinoor on your PC.Document the filter and include source code and explanation into your report.
- Create and compare the histograms for at least two different bandwidths and, if applicable, with or w/o filter. What are the reasons for the differences?
- How large is the offset of the yaw rate signal in the respective measurements?

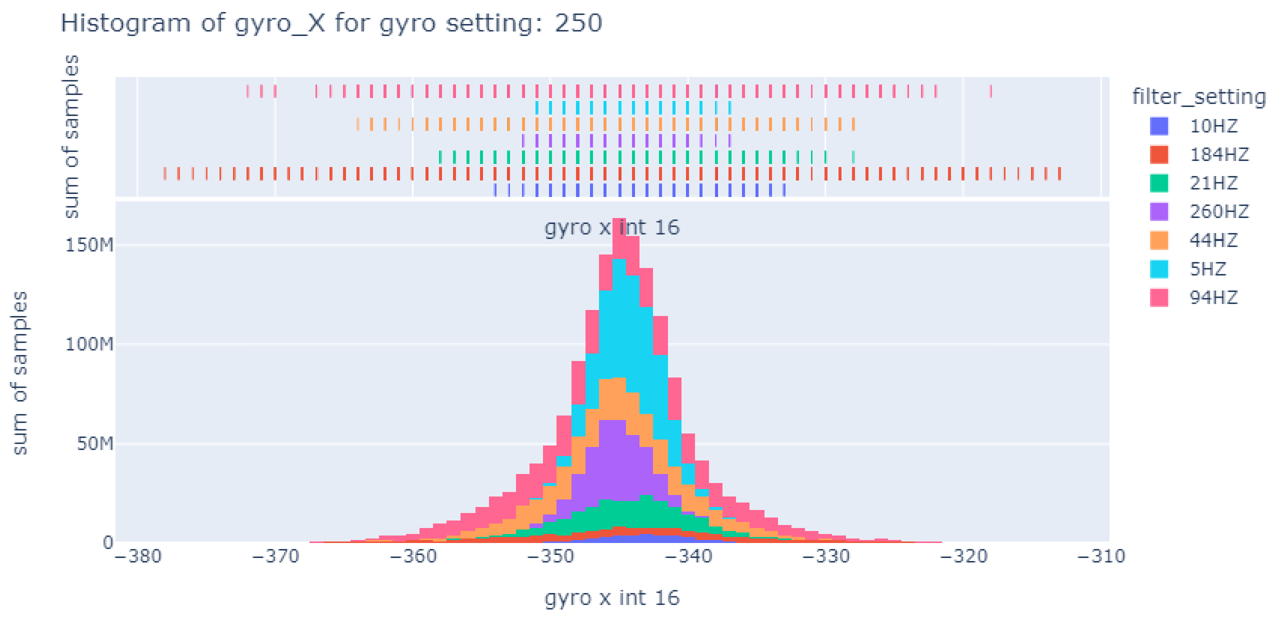


Fig 15 Histogram Gyroscope X 250dps Different Filters

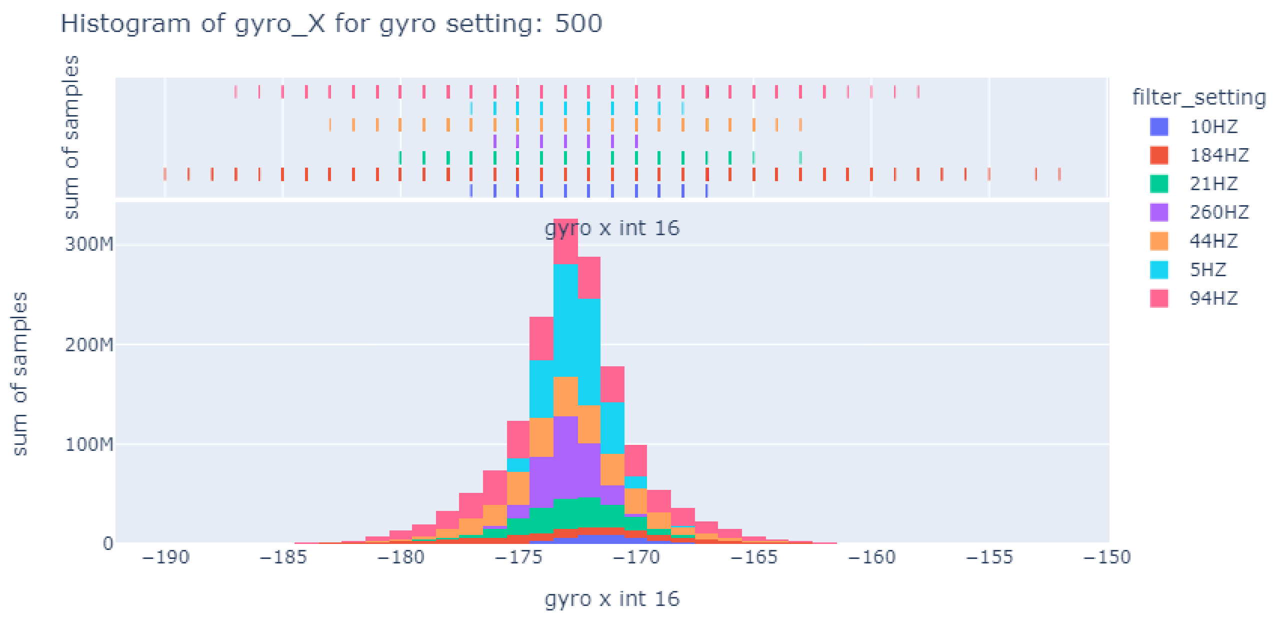


Fig 16 Histogram Gyroscope X 500dps Different Filters

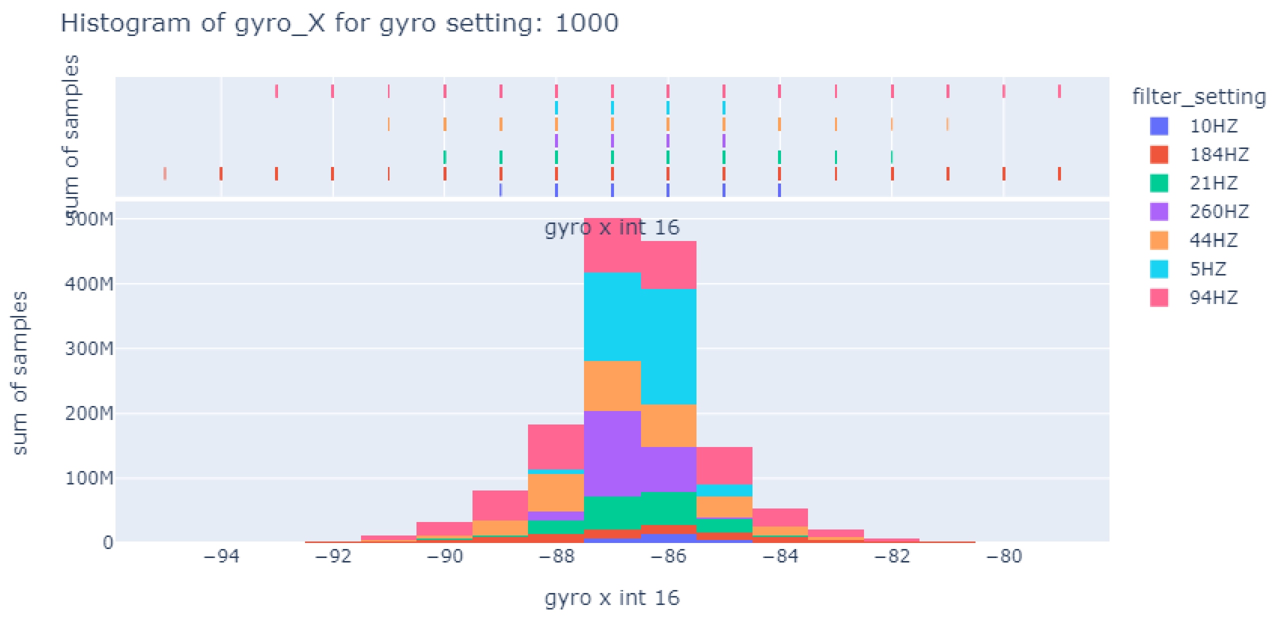


Fig 17 Histogram Gyroscope X 1000dps Different Filters

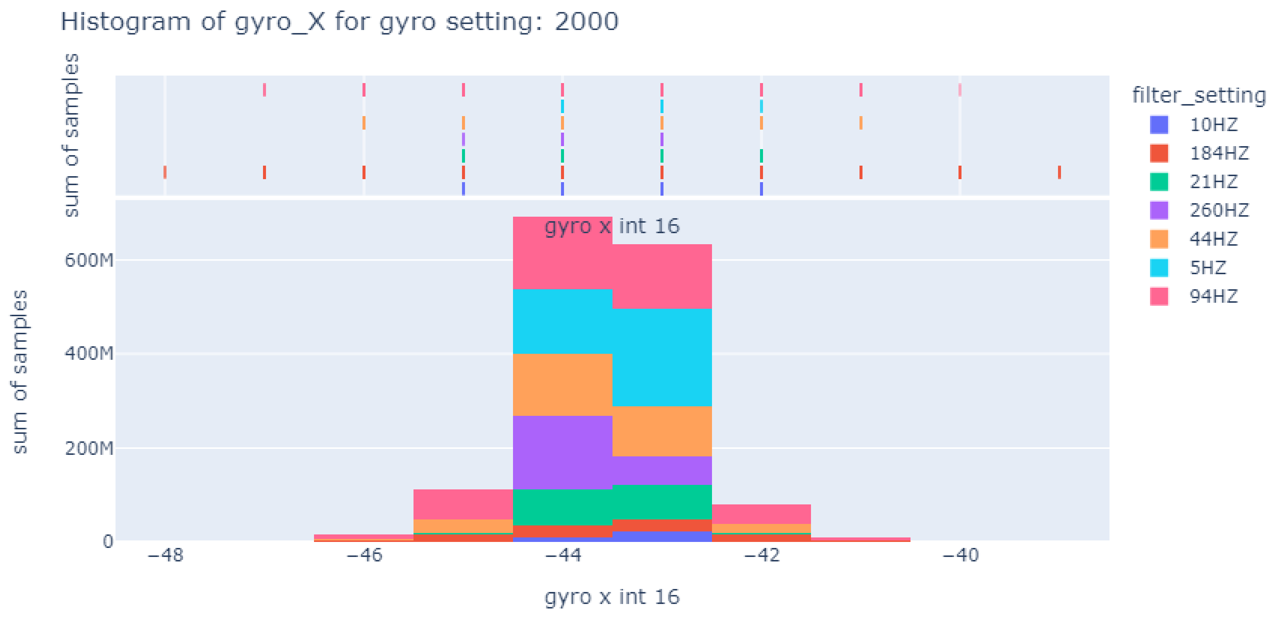


Fig 18 Histogram Gyroscope X 2000dps Different Filters

We again see no correlation in between the cutoff frequency on the low pass filter and the spread of the extreme values, as we originally expected. The register maps quotes the sensor as having a digital low pass filter. Perhaps the delta between the diferent values isnt big enough to trigger it and a more accurate test would be to move the sensor around with a know acceleration.

Part 6 Visualization with “Processing”

- Switch the setupto evaluating the data with Processing –for this you need the program „ArduinoProcessingMPU6050“ on the Arduino and „ProcessingMPU6050“withinProcessing

- Analyze the program for the Arduino. How does it work? How do you find out the correction values that need to be entered into the program?

The function calculate imu error capturres 200 samples and calculate the drift on x and y by averaging the samples and obtaining the x and y components of the drift vector.

```
void calculate_IMU_error() {
  // We can call this funtion in the setup section to calculate the accelerometer and gyro data
  // From here we will get the error values used in the above equations printed on the Serial Mc
  // Note that we should place the IMU flat in order to get the proper values, so that we then c
  // Read accelerometer values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
    // Sum all readings
    AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI));
    AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2) + pow((AccZ), 2))) * 180 /
    c++;
  }
  //Divide the sum by 200 to get the error value
  AccErrorX = AccErrorX / 200;
  AccErrorY = AccErrorY / 200;
  c = 0;
  // Read gyro values 200 times
  while (c < 200) {
    Wire.beginTransmission(MPU);
    Wire.write(0x43);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU, 6, true);
    GyroX = Wire.read() << 8 | Wire.read();
    GyroY = Wire.read() << 8 | Wire.read();
    GyroZ = Wire.read() << 8 | Wire.read();
    // Sum all readings
    GyroErrorX = GyroErrorX + (GyroX / 131.0);
    GyroErrorY = GyroErrorY + (GyroY / 131.0);
    GyroErrorZ = GyroErrorZ + (GyroZ / 131.0);
    c++;
  }
  //Divide the sum by 200 to get the error value
  GyroErrorX = GyroErrorX / 200;
  GyroErrorY = GyroErrorY / 200;
  GyroErrorZ = GyroErrorZ / 200;
  GyroErrorZ = GyroErrorZ + 0.01;

  // Print the error values on the Serial Monitor
  Serial.print("AccErrorX: ");
  Serial.println(AccErrorX);
  Serial.print("AccErrorY: ");
  Serial.println(AccErrorY);
  Serial.print("GyroErrorX: ");
  Serial.println(GyroErrorX);
```

```
    Serial.print("GyroErrorY: ");
    Serial.println(GyroErrorY);
    Serial.print("GyroErrorZ: ");
    Serial.println(GyroErrorZ);
  }
```

the value obtained is later use as a offset to correct the drift on the x and y axis.

```
    GyroX = GyroX - GyroErrorX ; // GyroErrorX ~(-0.56)
    GyroY = GyroY - GyroErrorY; // GyroErrorY ~(2)
    GyroZ = GyroZ - GyroErrorZ; // GyroErrorZ ~ (-0.8)
```

- Analyzethe program for Processing. What does this program do? How does it work?

The program for processing reads the data from the serial port, translate the 'cube' and display the yaw, pitch and roll values.

initiates the serial port

```
  void setup() {
    size (1024, 768, P3D);
    myPort = new Serial(this, "COM4", 19200); // starts the serial communication
    myPort.bufferUntil('\n');
    // logo = loadImage("UrbanMobilityLab.png");
    // logo2 = loadImage("HAW_Marke.png");
  }
```

reads the data on serial event

```
void serialEvent (Serial myPort) {
  // reads the data from the Serial Port up to the character '.' and puts it into the String var
  data = myPort.readStringUntil('\n');

  // if you got any bytes other than the linefeed:
  if (data != null) {
    data = trim(data);
    // split the string at "/"
    String items[] = split(data, '/');
    if (items.length > 1) {

      //--- Roll,Pitch in degrees
      roll = float(items[0]);
      pitch = float(items[1]);
      yaw = float(items[2]);
    }
  }
}
```

draws the cube and the text, call on every frame

```
void draw() {
  translate(width/2, height/2, 0);
  background(233);
  textSize(22);
  text("Roll: " + int(roll) + "     Pitch: " + int(pitch), -100, 265);

  // Rotate the object
  rotateX(radians(-pitch));
  rotateZ(radians(roll));
  rotateY(radians(yaw));

  // 3D 0bject
  textSize(20);
  fill(0, 76, 153);
  box (500, 40, 200); // Draw box
  textSize(25);
  fill(255, 255, 255);
  text("HAW Hamburg", -183, 10, 101);
  //image(logo,0,-300,400,400);
  //image(logo2,-200,0);
  delay(10);
  //println("ypr:\t" + angleX + "\t" + angleY); // Print the values to check whether we are gett
}
```

- Try itout: Move the sensor and watch the screen. How do you know that your sensor is not yet perfectly calibrated?

if the sensor drifts it means that the sensor is not yet perfectly calibrated as the continous integration of the gyroscope values and its inherent error will add up over time generating a drift.

- Document your results with a screendump in your lab report.

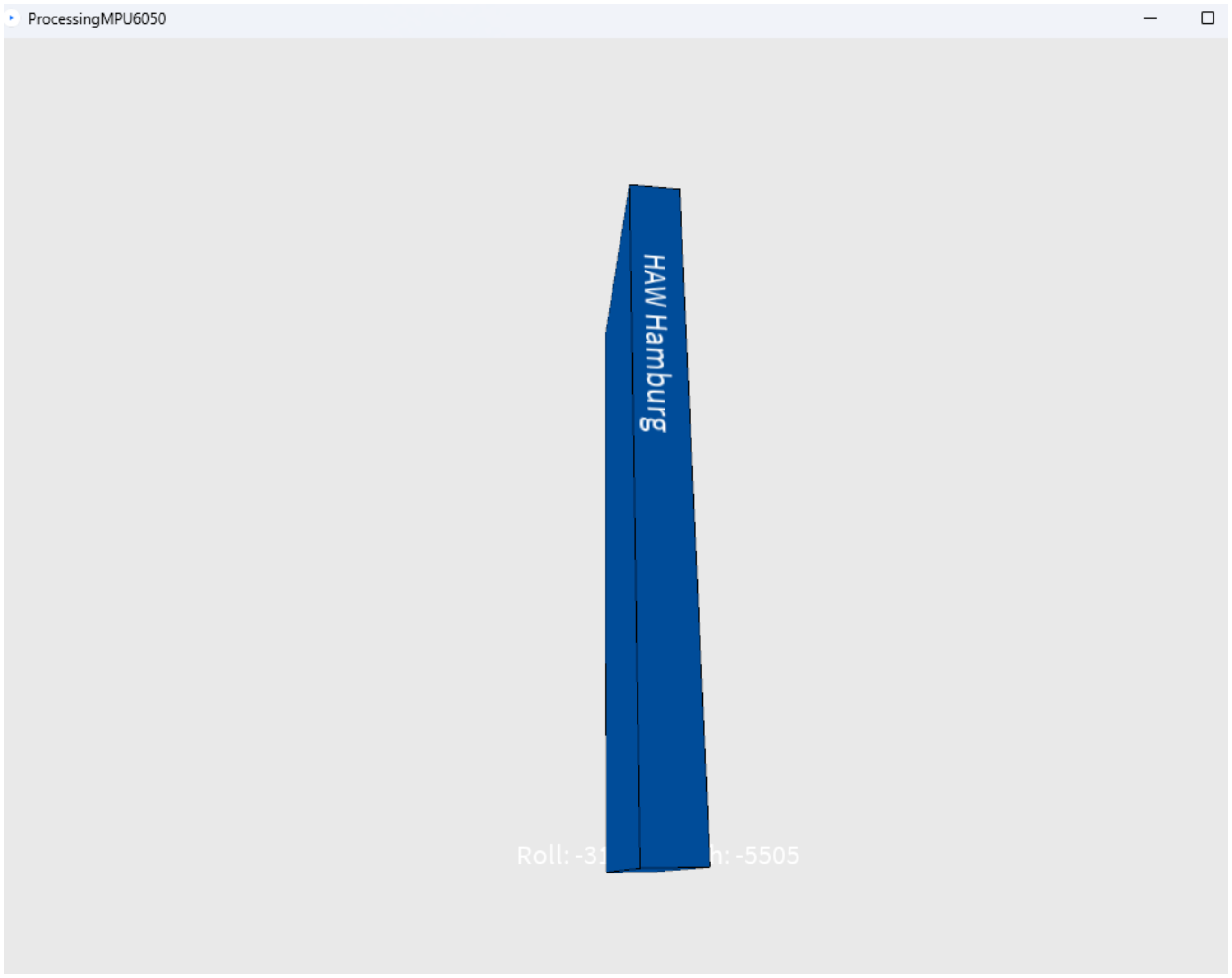


Fig 19 Processing

References

- MPU6050