Table of Contents

Introduction

This project aims to create a real-time vibration analysis system capable of collecting and analyzing vibration data to provide useful insights. The system uses accelerometer sensor (MPU 6050), and processes the raw input data to produce a desired output. A command line interface (CLI) is used to display the output data for easy monitoring.

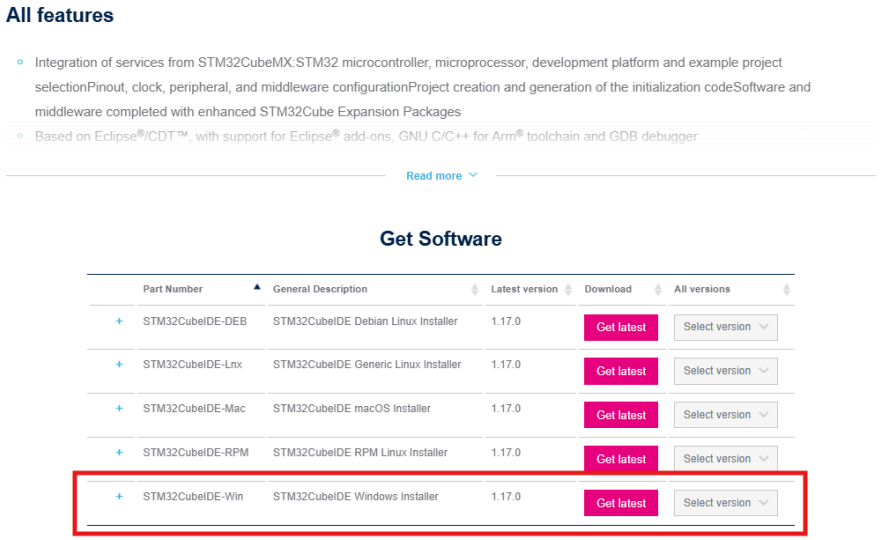
Components

1. Breadboard
2. STM32F411CEU6 (black pill)
3. ST- link/V2
4. Jumper Wire
5. MPU 6050
6. LED

Steps

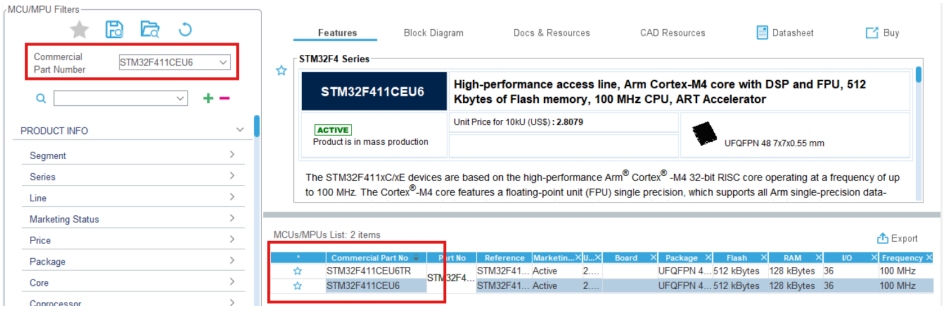
1. Download necessary softwares: STM32CubeIDE

   - Go to the ST official website. When downloading this for the first time, users need to manually create a MyST account with their email. After that, the download link will be sent to the email inbox. Simply click the link, and it will start to download automatically.



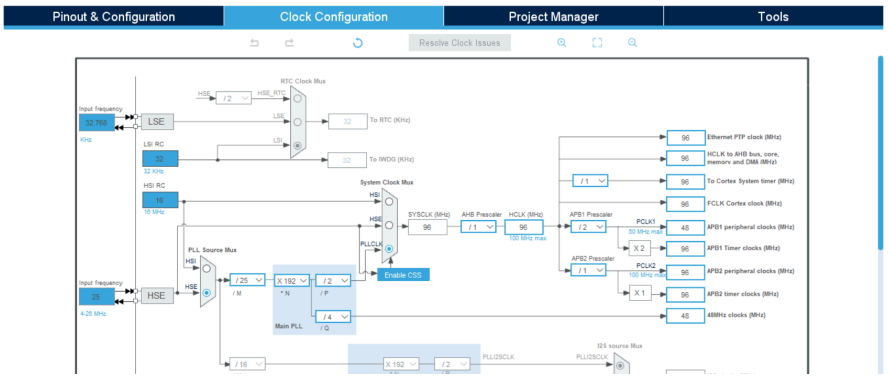
2. Manually setup STM32 blackpill (STM32F411CEU6) in STM32CubeIDE.

   i. Click on "Create a New STM32 project", enter "STM32F411CEU6" in Commercial Part Number. And select it MCUs/MPUs List.



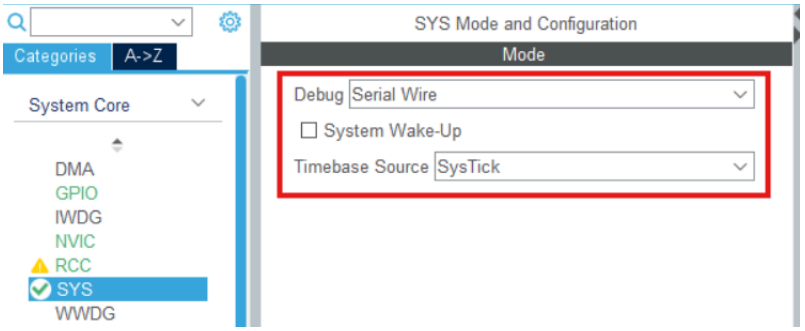
   ii. Enter any project name, and then select the file location. After that, click Finish.

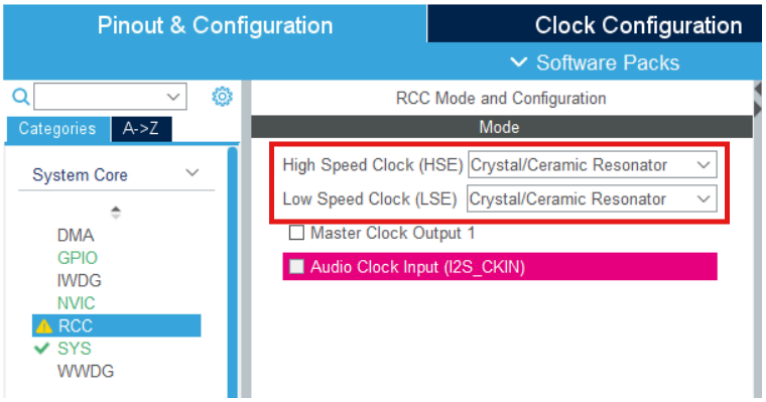
iii.Go to Clock Configurations tab and configure the settings as below



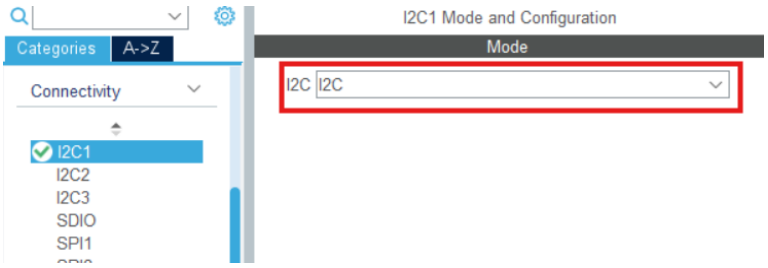
iv.Go to Pinout & Configurations tab and configure the settings as below:

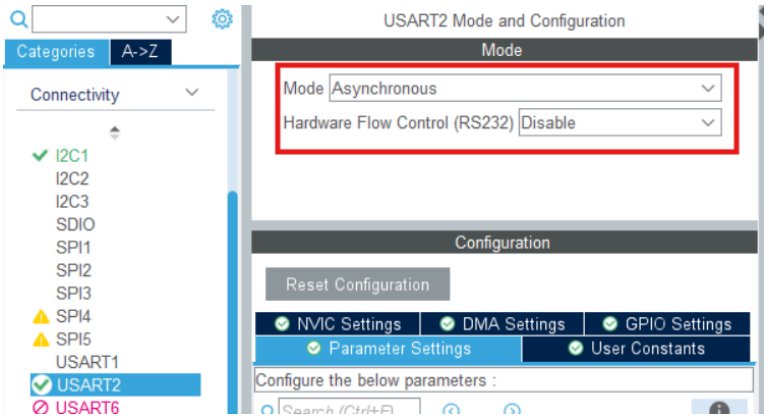
-At system core



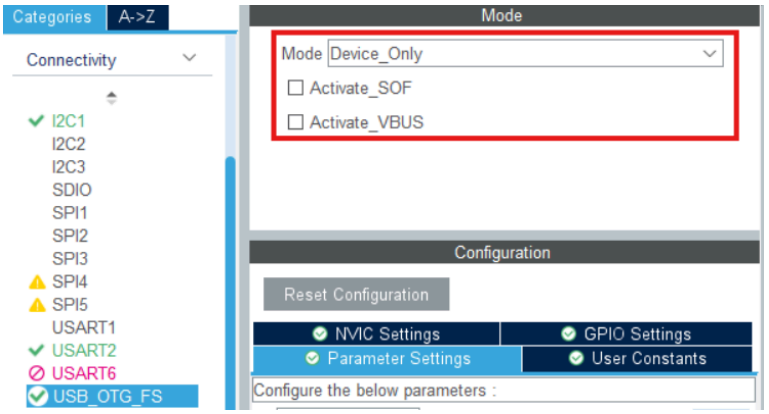


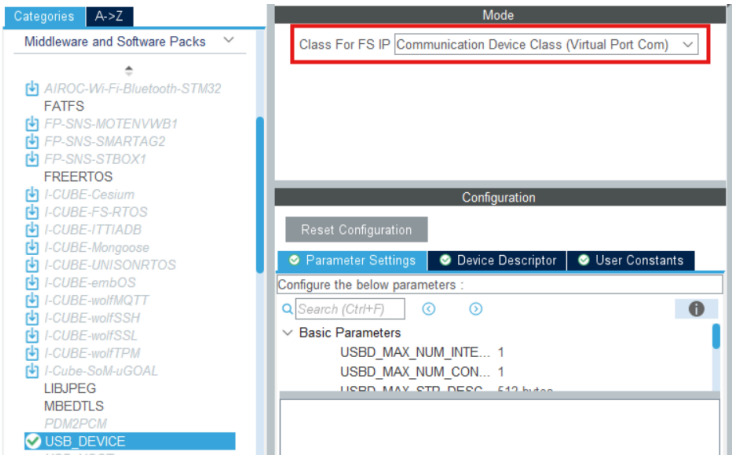
-At connectivity:



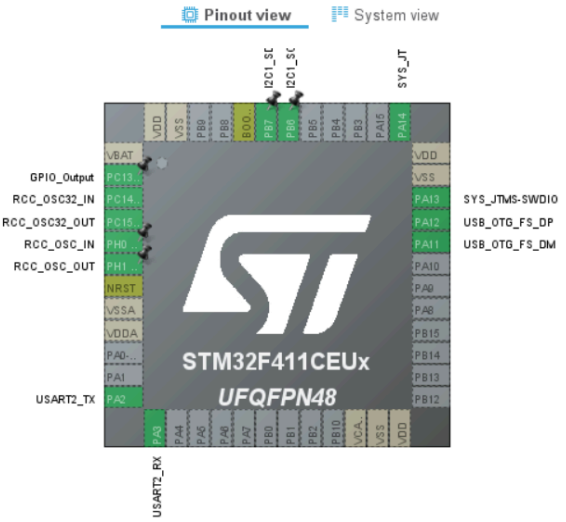


-At Middleware & Software Pack:

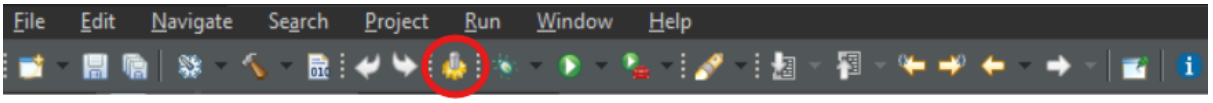




v. In the end, the pinout view would look like this.

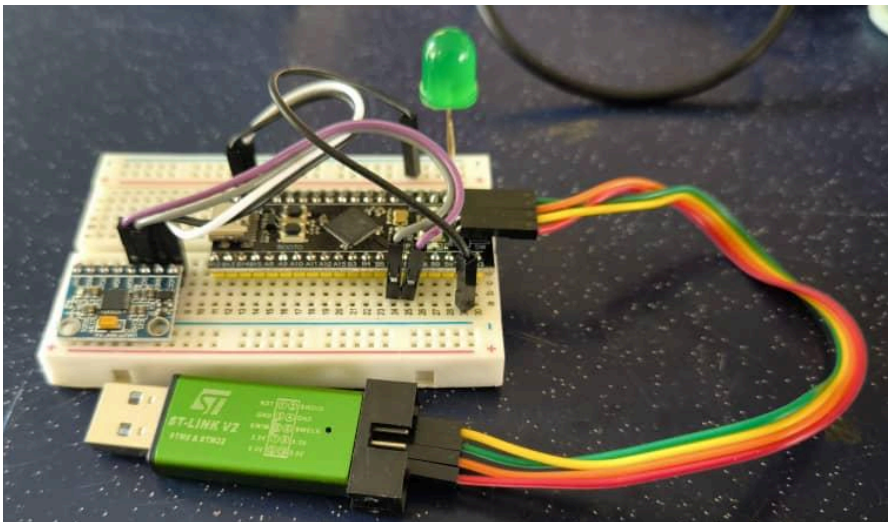


vi. Finally, generate the code. A main.c file will be generated with all the settings configured. Click the code generation button (Alt + K)

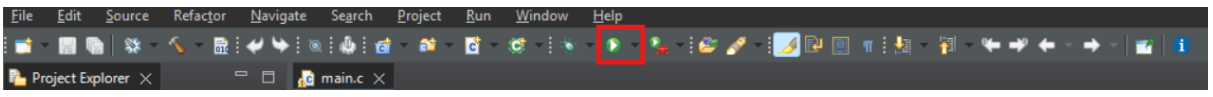


3. Run and Flash.

- Simply copy all the code in the repo and paste it all in the main.c file.
- Connect the Black Pill with ST-Link V2 according to its pinouts.



- Connect ST-Link V2 with PC.
- Click the run button to build the project and download to Black Pill.



Schematic Diagram

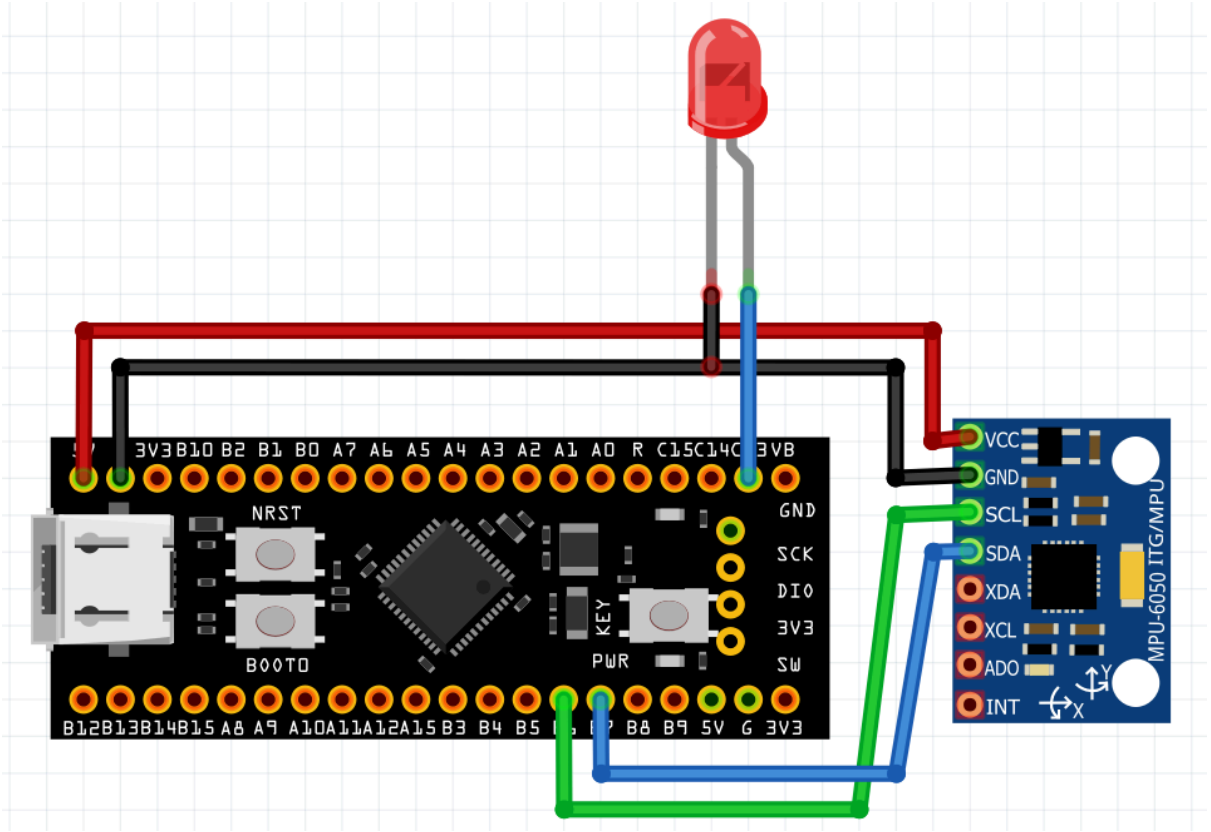


Figure 1.1: schematic diagram of real time vibration analysis

Flow Chart Diagram

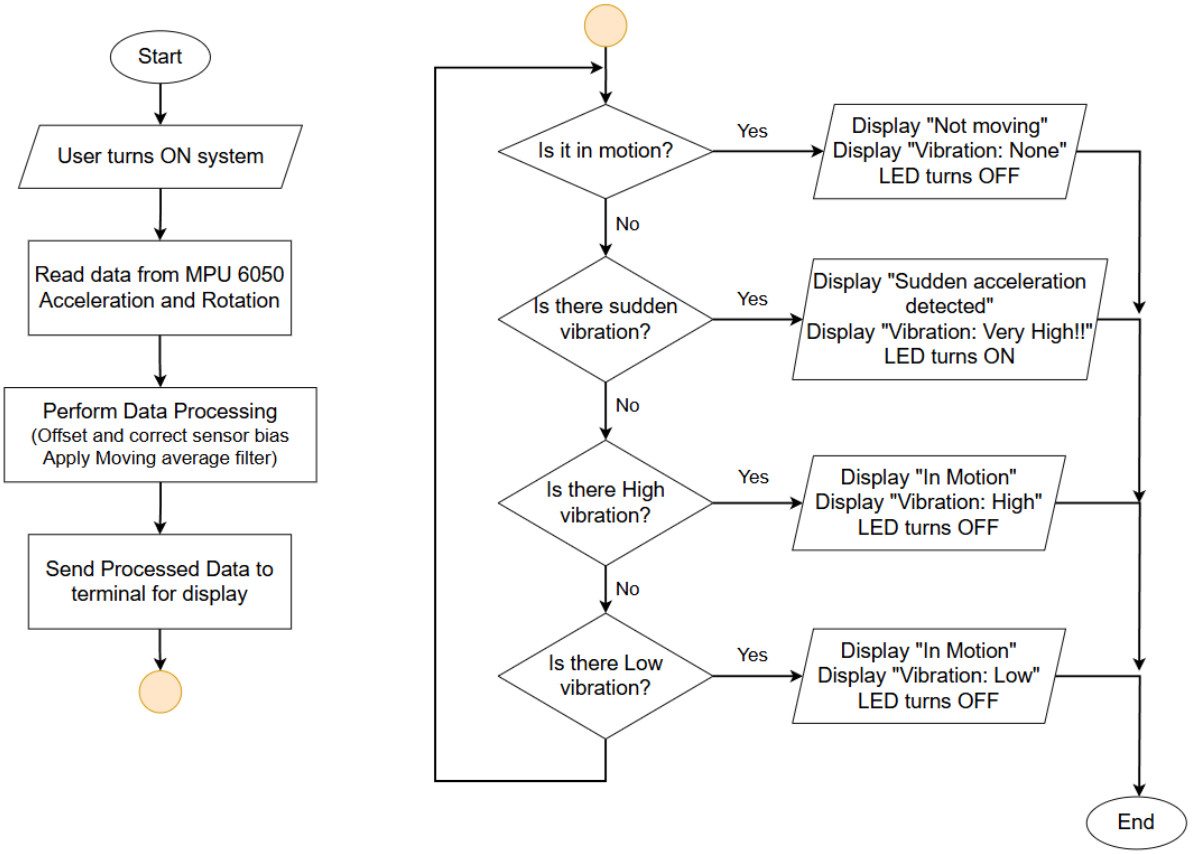


Figure 1.2: Flow chart diagram of real time vibration analysis

Code:

-The following link contains the documentation and codebase for the coding of this project:

Real Time Vibration Analysis Documentation

-The zip file and 7zip file can be downloaded directly here:

Real Tine Vibration Analysis Releases

Full Code:

```
// Include add needed libraries
#include "main.h"
#include "usb_device.h"
#include <stdio.h>

// Initialize I2C communication
I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart2;

// Initialize private variables
int x = 0;
char msg[20];
extern uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len);

// Private function
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART2_UART_Init(void);

// Define Address and registers
#define MPU6050_ADDR 0xD0
#define SMPLRT_DIV_REG 0x19
#define GYRO_CONFIG_REG 0x1B
#define ACCEL_CONFIG_REG 0x1C
#define ACCEL_XOUT_H_REG 0x3B
#define TEMP_OUT_H_REG 0x41
#define GYRO_XOUT_H_REG 0x43
#define PWR_MGMT_1_REG 0x6B
#define WHO_AM_I_REG 0x75


int16_t Accel_X_RAW = 0;
int16_t Accel_Y_RAW = 0;
int16_t Accel_Z_RAW = 0;

int16_t Gyro_X_RAW = 0;
int16_t Gyro_Y_RAW = 0;
```

```
int16_t Gyro_Z_RAW = 0;

float Ax, Ay, Az, Gx, Gy, Gz;

void MPU6050_Init (void)
{
	uint8_t check;
	uint8_t Data;

	// check device ID WHO_AM_I
	HAL_I2C_Mem_Read (&hi2c1, MPU6050_ADDR, 0x75, 1, &check, 1, 1000);
	if (check == 0x68)
	{

		Data = 0;
		HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, 0x6B, 1,&Data, 1, 1000);

		// Set DATA RATE of 1KHz by writing SMPLRT_DIV register
		Data = 0x07;
		HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, 0x19, 1, &Data, 1, 1000);

		// Set Gyroscopic configuration in GYRO_CONFIG Register
		Data = 0x00;  // XG_ST=0,YG_ST=0,ZG_ST=0, FS_SEL=0 -> ± 250 °/s
		HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, 0x1B, 1, &Data, 1, 1000);

		// Set accelerometer configuration in ACCEL_CONFIG Register
		Data = 0x00;  // XA_ST=0,YA_ST=0,ZA_ST=0, FS_SEL=0 -> ± 2g
		HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, 0x1C, 1, &Data, 1, 1000);
	}
}


void MPU6050_Read_Accel (void)
{
	uint8_t Rec_Data[6];

	// Read 6 BYTES of data starting from ACCEL_XOUT_H (0x3B) register
	HAL_I2C_Mem_Read (&hi2c1, MPU6050_ADDR, 0x3B, 1, Rec_Data, 6, 1000);

	Accel_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data [1]);
	Accel_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data [3]);
	Accel_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data [5]);

	/*** convert the RAW values into acceleration in 'g'
	     we have to divide according to the Full scale value set in FS_SEL
	     I have configured FS_SEL = 0. So I am dividing by 16384.0
	     for more details check ACCEL_CONFIG Register              ****/

	// Set values and calibrate with offset
	Ax = ((float)Accel_X_RAW/16384.0) + 0.14;
	Ay = ((float)Accel_Y_RAW/16384.0) - 0.02;
```

```
	Az = ((float)Accel_Z_RAW/16384.0) - 0.94;
}


// Define previous values and threshold for zero motion
static float previous_Gx = 0.0, previous_Gy = 0.0, previous_Gz = 0.0;
#define GYRO_THRESHOLD 0.1  // Define a threshold for zero motion
void MPU6050_Read_Gyro (void)
{
	uint8_t Rec_Data[6];

	// Read 6 BYTES of data starting from GYRO_XOUT_H register
	HAL_I2C_Mem_Read (&hi2c1, MPU6050_ADDR, 0x43, 1, Rec_Data, 6, 1000);

	Gyro_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data [1]);
	Gyro_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data [3]);
	Gyro_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data [5]);

	/*** convert the RAW values into dps (°/s)
	     we have to divide according to the Full scale value set in FS_SEL
	     I have configured FS_SEL = 0. So I am dividing by 131.0
	     for more details check GYRO_CONFIG Register              ****/
	// Convert RAW values into degrees per second (dps)
	Gx = ((float)Gyro_X_RAW/131.0);
	Gy = ((float)Gyro_Y_RAW/131.0);
	Gz = ((float)Gyro_Z_RAW/131.0);

	// Apply threshold to ignore small gyro values (no motion)
	if (fabs(Gx) < GYRO_THRESHOLD) Gx = 0.0;
	if (fabs(Gy) < GYRO_THRESHOLD) Gy = 0.0;
	if (fabs(Gz) < GYRO_THRESHOLD) Gz = 0.0;

	// Apply moving average filter for smoothing
	Gx = (Gx + previous_Gx) / 2.0;
	Gy = (Gy + previous_Gy) / 2.0;
	Gz = (Gz + previous_Gz) / 2.0;

	// Update previous values
	previous_Gx = Gx;
	previous_Gy = Gy;
	previous_Gz = Gz;
}

#define VIBRATION_LOW_THRESHOLD 0.5  // Threshold for low vibration (dps)
#define VIBRATION_HIGH_THRESHOLD 1.0  // Threshold for high vibration (dps)
int main(void)
{
// Initialize HAL commands
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
```

```
MX_I2C1_Init();
MX_USB_DEVICE_Init();
MX_USART2_UART_Init();
MPU6050_Init();
HAL_StatusTypeDef ret = HAL_I2C_IsDeviceReady(&hi2c1, (0b1101000 <<1) + 0, 1,
100);

  while (1){
       MPU6050_Read_Accel();
       MPU6050_Read_Gyro();

       char msg[200];
       sprintf(msg, "Ax=%.2fg Ay=%.2fg Az=%.2fg \r\n\nGx=%.2f  Gy=%.2f  Gz=%.2f
\r\n\n\n\n", Ax,Ay,Az,Gx,Gy,Gz);
       CDC_Transmit_FS((uint8_t*)msg, strlen(msg));
       HAL_Delay(200);

       ////////////////////////////////////////////
       float prev_Ax = 0.0, prev_Ay = 0.0, prev_Az = 0.0;  // Store previous
acceleration values
       float threshold = 1.8;  // Define a threshold for sudden acceleration

       // Calculate the difference (zero-crossing method)
       float delta_Ax = Ax - prev_Ax;
       float delta_Ay = Ay - prev_Ay;
       float delta_Az = Az - prev_Az;

       // Check for sudden acceleration
       if (fabs(delta_Ax) > threshold || fabs(delta_Ay) > threshold ||
fabs(delta_Az) > threshold){
              char alert_msg[] = "!!Sudden acceleration detected\r\nVibration:
VERY HIGH!!\r\n\n";
              CDC_Transmit_FS((uint8_t*)alert_msg, strlen(alert_msg));
              HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);  // Turn ON
external LED, off for onboard
              HAL_Delay(3000);
       }
       else{
              // Calculate total vibration magnitude
              float vibration_magnitude = sqrt(Ax * Ax + Ay * Ay + Az * Az);

              // Check if there is no movement (vibration magnitude near 0)
              if (vibration_magnitude < GYRO_THRESHOLD) {
                     char no_movement_msg[] = "Not Moving\r\nVibration:
None\r\n\n";
                     CDC_Transmit_FS((uint8_t*)no_movement_msg,
strlen(no_movement_msg));
                     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);  //
Turn OFF external LED, on for onboard
              }
              // Check for low vibration
```

```
            else if (vibration_magnitude < VIBRATION_LOW_THRESHOLD) {
                  char low_vibration_msg[] = "In Motion\r\nVibration:
LOW\r\n\n";
                  CDC_Transmit_FS((uint8_t*)low_vibration_msg,
strlen(low_vibration_msg));
                  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);  //
Turn OFF external LED, on for onboard
            }
            // Check for high vibration
            else if (vibration_magnitude >= VIBRATION_HIGH_THRESHOLD) {
                  char high_vibration_msg[] = "Moving\r\nVibration:
HIGH\r\n\n";
                  CDC_Transmit_FS((uint8_t*)high_vibration_msg,
strlen(high_vibration_msg));
                  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);  //
Turn OFF external LED, on for onboard
            }
            HAL_Delay(100);
      }
      // Update previous values
      prev_Ax = Ax;
      prev_Ay = Ay;
      prev_Az = Az;
  }
}
```

Code Explanation:

1. Reading Acceleration Data:

```
// Read 6 BYTES of data starting from ACCEL_XOUT_H (0x3B) register
HAL_I2C_Mem_Read (&hi2c1, MPU6050_ADDR, 0x3B, 1, Rec_Data, 6, 1000);

Accel_X_RAW = (int16_t)(Rec_Data[0] << 8 | Rec_Data [1]);
Accel_Y_RAW = (int16_t)(Rec_Data[2] << 8 | Rec_Data [3]);
Accel_Z_RAW = (int16_t)(Rec_Data[4] << 8 | Rec_Data [5]);

/*** convert the RAW values into acceleration in 'g'

// Set values and calibrate with offset
Ax = ((float)Accel_X_RAW/16384.0) + 0.14;
Ay = ((float)Accel_Y_RAW/16384.0) - 0.02;
Az = ((float)Accel_Z_RAW/16384.0) - 0.94;
```

   - Reads 6 bytes of accelerometer data from register 0x3B of the MPU6050 using I2C.
   - The MPU6050 default FS_SEL = 0 (Full Scale ±2g), so divide by 16384 to get values in g.
   - Offsets (+0.14, -0.02, -0.94) correct sensor biases to improve accuracy.

2. Reading Gyroscope data:

```
/*** convert the RAW values into dps (°/s)
// Convert RAW values into degrees per second (dps)
Gx = ((float)Gyro_X_RAW/131.0);
Gy = ((float)Gyro_Y_RAW/131.0);
Gz = ((float)Gyro_Z_RAW/131.0);

// Apply threshold to ignore small gyro values (no motion)
if (fabs(Gx) < GYRO_THRESHOLD) Gx = 0.0;
if (fabs(Gy) < GYRO_THRESHOLD) Gy = 0.0;
if (fabs(Gz) < GYRO_THRESHOLD) Gz = 0.0;

// Apply moving average filter for smoothing
Gx = (Gx + previous_Gx) / 2.0;
Gy = (Gy + previous_Gy) / 2.0;
Gz = (Gz + previous_Gz) / 2.0;

// Update previous values
previous_Gx = Gx;
previous_Gy = Gy;
previous_Gz = Gz;
```

   - Convert Raw Gyro Data to Degrees per Second (°/s)
   - The MPU6050 gyroscope default FS_SEL = 0 (Full Scale ±250°/s), so divide by 131 to convert raw values to °/s.
   - `fabs()` gets the absolute value.

- If the gyro values are below **GYRO_THRESHOLD**, set them to **0** (eliminates noise when stationary).
- Apply Moving Average Filter (Smoothing). Averages the current and previous values to reduce sudden fluctuations.

3. Transmitting data to terminal:

```
char msg[200];
sprintf(msg, "Ax=%.2fg Ay=%.2fg Az=%.2fg \r\n\nGx=%.2f  Gy=%.2f  Gz=%.2f \r\n\n\n\n", Ax,Ay,Az,Gx,Gy,Gz);
CDC_Transmit_FS((uint8_t*)msg, strlen(msg));
HAL_Delay(200);
```

- Create a Character Array for Message Storage
- `sprintf()` formats the accelerometer (`Ax, Ay, Az`) and gyroscope (`Gx, Gy, Gz`) values into a human-readable string.
- `%.2f` ensures values are displayed **with 2 decimal places**.
- `\r\n\n` adds **line breaks** for better readability.
- Sends the formatted message to the PC via **USB Virtual COM Port** (CDC - Communication Device Class).
- Sends out data at 200ms interval

4. Check for sudden acceleration:

```
////////////////////////////////////////////
float prev_Ax = 0.0, prev_Ay = 0.0, prev_Az = 0.0;  // Store previous acceleration values
float threshold = 1.8;  // Define a threshold for sudden acceleration

// Calculate the difference (zero-crossing method)
float delta_Ax = Ax - prev_Ax;
float delta_Ay = Ay - prev_Ay;
float delta_Az = Az - prev_Az;

// Check for sudden acceleration
if (fabs(delta_Ax) > threshold || fabs(delta_Ay) > threshold || fabs(delta_Az) > threshold){
    char alert_msg[] = "!!Sudden acceleration detected\r\nVibration: VERY HIGH!!\r\n\n";
    CDC_Transmit_FS((uint8_t*)alert_msg, strlen(alert_msg));
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);  // Turn ON external LED, off for onboard
    HAL_Delay(3000);
}
```

- Initialize Previous Acceleration Values and Threshold
- `threshold = 1.8` Defines the sensitivity for detecting sudden acceleration
- Calculate Acceleration Change (Zero-Crossing Method)
- Check for Sudden Acceleration, with `fabs()` to get the absolute value of acceleration differences
- Send Warning Message via USB
- Turn ON external LED to Indicate Alert

Results:

-The following image is the Real Time Vibration device built:

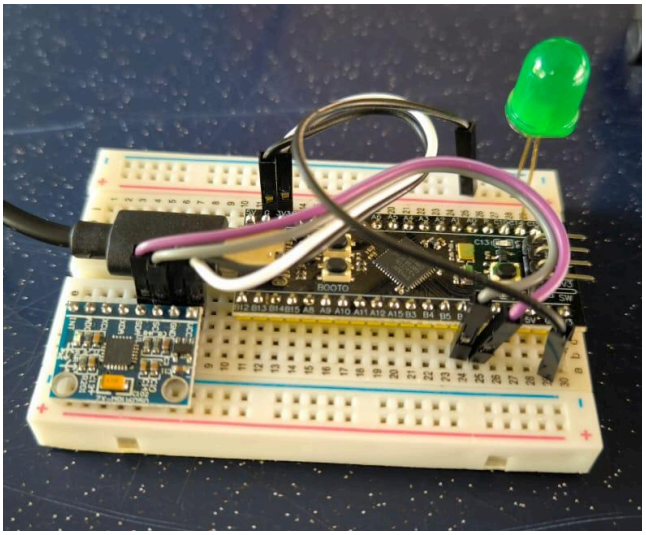


Figure 2.1: Real Time Vibration device in normal state

-The green LED light will light up when the device is shaken beyond a specific threshold.

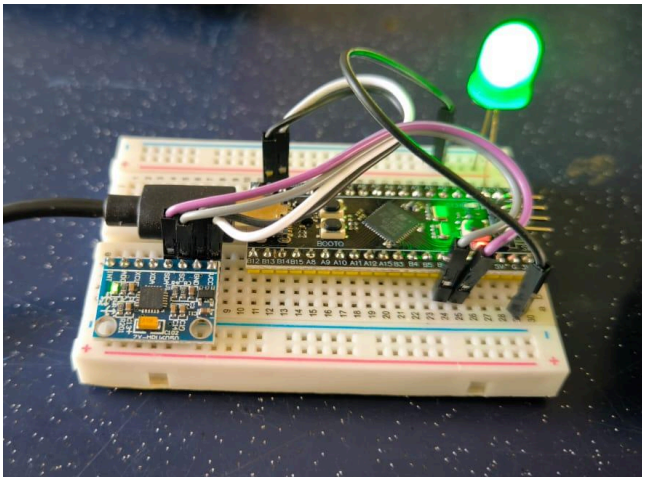


Figure 2.2: Real Time VIbration device when it is shaken sufficiently

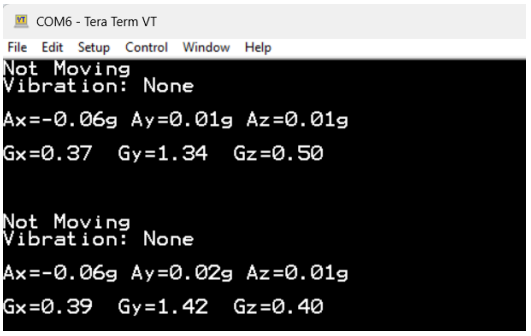


Figure 2.3: Coding result when the device is not moved

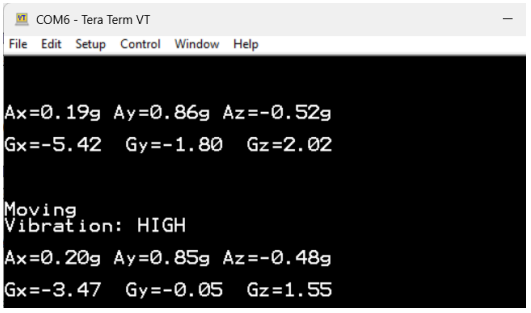


Figure 2.3: Coding result when the device is gently shaken

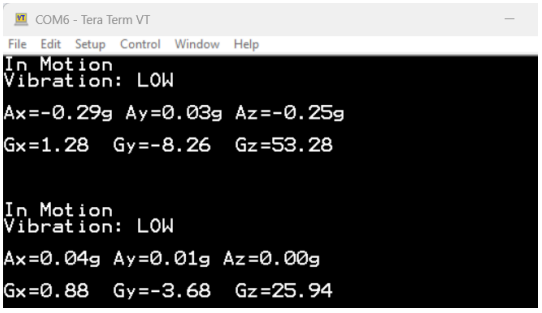


Figure 2.3: Coding result when the device is shaken vigorously

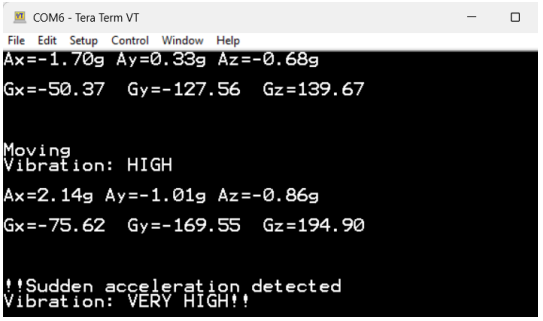


Figure 2.3: Coding result when the device experienced a sudden acceleration

Demonstration Video Link:

demo video.mp4

Practical application:

This project has practical applications in vibration monitoring and motion detection, making it useful in various fields such as detecting vibrations in machinery, monitoring structural integrity, industrial machinery diagnostics, and analyzing dynamic motions in robotics and vehicles.

By analyzing acceleration and gyro data, the system can detect sudden movements, vibrations, or instability, allowing for real-time alerts and preventive maintenance. In automotive applications, it can help in accident detection or stability control, while in industrial settings, it can be used to monitor equipment health and detect mechanical faults before failures occur.

Stm32 Black Pill & MPU 6050:

The STM32 Black Pill microcontroller serves as the core processing unit in this project, handling sensor data input and output. It processes acceleration and gyroscope readings from the MPU6050, a 6-axis motion tracking sensor that provides real-time motion and vibration data. Using I2C communication, the Black Pill retrieves sensor measurements, analyzes the data, and transmits the processed data to a terminal for display.

Prompting tips:

These prompts were input into the settings of GPTs to enhance the code generation process.

ChatGPT (OpenAI):
"Keep responses short and concise. When I request code, provide only the code. When I ask for corrections, just correct the code. No explanations unless I specifically request them. I'm building a real-time vibration analysis system using the STM32 Black Pill with STM32CubeIDE and an MPU6050 sensor."

Gemini (Google):
“Real-time vibration analysis with STM32 Black Pill. Using the MPU6050 (accelerometer and gyroscope data) and STM32CubeIDE. Provide code directly. Correct code upon request. Troubleshoot errors with solutions. Keep responses concise and code-focused unless I specifically request explanations.”
To start with prompts:
“Code to initialize the MPU6050 using I2C communication in STM32CubeIDE. I need to read the accelerometer and gyroscope data.”

DeepSeek (DeepSeek Artificial Intelligence Co):
“Please keep all responses concise and to the point. Follow these guidelines:

1. When I request code, provide only the code without explanations.
2. When I ask for corrections to code, provide only the corrected code unless I specify otherwise.
3. Avoid explaining each part of the code unless I explicitly ask for an explanation.
4. If I encounter an error, help me troubleshoot it by providing possible solutions.

I am working on a real-time vibration analysis project using the STM32 Black Pill microcontroller and STM32CubeIDE. I will be using the MPU6050 sensor for this project. Please assist me with code, corrections, and troubleshooting as needed.”

Perplexity (Perplexity AI):

"As an experienced STM32 embedded systems engineer, provide concise assistance. I'm building a real-time vibration analysis system using an STM32 Black Pill and an MPU6050 sensor, programming in STM32CubeIDE. When I request code, provide only the code. For code corrections, provide only the corrected code. Unless specified, skip detailed explanations. When I encounter an error, help me troubleshoot it with potential solutions. Assume I have a basic understanding."

Limitations

1. Using HAL commands instead of bare-metal:
   HAL functions add extra processing time, making them less efficient for real-time applications. At the same time, HAL functions are slower than direct register manipulation.
2. Lack of casing:
   Casing can protect the device from dust, debris, and physical damage. Without a fixed enclosure, sensor readings may be inconsistent due to unintended movements.
3. Lack of battery:
   Lacks the portability of battery powered systems. This project can only be used when connected with a USB-C cable. The system requires a constant power source, restricting its mobility. Any power loss or disconnection can disrupt real-time analysis.

Future upgrade

1. Add a battery as power source:

   Adding a battery to power the project can improve portability greatly. Hence, the system can operate independently without being tethered to a power source. This enables the project to be used in remote or mobile applications without relying on external power.

2. Use 3D printed casing as enclosure:

   An enclosure can help prevent components from environmental exposure, such as dust, debris and moisture. Additionally, the enclosure can provide mounting points for the project, so that the project can be secured on an object.

3. Interface with additional components:

   Add extra components to enhance the usability of the project. For example, adding an LCD display to provide features such as live data display without using CLI. A buzzer would also be useful, such as alerting the user when sudden acceleration is detected. Finally, a relay module can be integrated into the project. For instance, when sudden acceleration is detected, the relay module can turn On/Off a peripheral specified by the user.

Conclusion

In conclusion, the Real-Time Vibration Analysis project successfully demonstrated how an STM32 Black Pill microcontroller and an MPU6050 sensor can be used to monitor vibrations in real time. By collecting and analyzing acceleration and gyroscope data, the system was able to detect movement and provide useful feedback. The use of STM32CubeIDE allowed for efficient programming and data processing, while Tera Term displays data through a serial monitor.

However, there were some limitations. The project relied on HAL commands, which are less efficient than direct register manipulation. The absence of an enclosure made the setup vulnerable to external factors, and the lack of a battery limited its portability. To improve the system, some future upgrades could include a rechargeable battery, a protective casing, and additional features for expanded functionality.

Despite these challenges, the project shows promise for practical applications in industrial monitoring, structural health assessment, and motion detection systems.

References (& video)

[1] STMicroelectronics, "Getting started with I2C," *STM32 MCU Wiki*. Available: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_I2C. (27 September 2024).

[2] STM32 World, "Black Pill - Pinout," *STM32 World Wiki*. Available: https://stm32world.com/wiki/Black_Pill#Pinout. (2 February 2025).

[3] Controllerstech, "STM32 HAL I2C and MPU6050 IMU," *YouTube*. [Online]. Available: https://www.youtube.com/watch?v=P7a6qxacnO4. (3 May 2022).

[4] Controllerstech, "How to Interface MPU6050 (GY-521) with STM32," *Controllerstech*. Available:https://controllerstech.com/how-to-interface-mpu6050-gy-521-with-stm32/ (11 June 2019)

[5] Bennett Notes, "STM32 Blackpill with STM32CubeIDE and USB Serial," *Bennett Notes*. Available: https://www.bennettnotes.com/notes/stm32-blackpill-with-stmcubeide-usb-serial/. (21 August 2021)

[6] Electronics Hub, "MPU6050 Arduino Tutorial for Beginners," *YouTube*. [Online]. Available: https://www.youtube.com/watch?v=SVI_NldMjlE. (15 May 2020)

[7] InvenSense, "MPU-6000 and MPU-6050 Product Specification," *Datasheet*. [Online]. Available: https://download.mikroe.com/documents/datasheets/MPU-6000_datasheet.pdf. (19 August 2013)