

## Lab 4 Report

### 4.1:

I [used ChatGPT](#) to generate the code for utilizing the IMU. It took a few tries to get an appropriate answer, at one point it used memcpy to and buffers to read the data from the IMU, which was unnecessarily complicated. With the generated code, I adjusted the GPIO pins and decided that any accelerometer values above 150 or below -150, would activate the printout.

### 4.2:

I had a few difficulties with this part. There were two examples in the esp-idf/examples/bluetooth folder that I felt like were promising. The first was [“esp\\_hid\\_device”](#), which I managed to get its volume adjustment capabilities working. I tried to use another under [bluetooth/bluedroid/ble/ble\\_hid\\_device\\_demo](#), which I could connect to via bluetooth, but it didn't seem to adjust my volume as it described in the README, so I opted for the first option. Although it describes in the documentation to have mouse functionality, getting said mouse functionality to work proved a challenge. Switching over to nimBLE allowed me to change the examples functionality via menuconfig, however my computer would constantly disconnect and reconnect with the device with error 531. I tried to debug and figure out what the problem was, but after speaking with a TA, they informed me that using the second example would be easier. After explaining that it didn't work my laptop, we worked together and found that for some reason, my laptop could connect to the esp32c3 via bluetooth using the example, but nothing I implemented using it would function. But if a different laptop or device connected to it, it functioned perfectly fine. In order to be able to test my code, I set up xfce4 on my raspberry pi along with blueman to handle bluetooth connectivity. With this information, I used the second example and implemented mouse movement using the esp\_hidd\_send\_mouse\_value function.

### 4.3:

After completing 4.2, combining the two was fairly simple. I copied over the files from 4.1 and 4.2 into one folder. I removed app\_main from 4.1 and moved the code into functions that would be called from the 4.2 code instead. Some additional changes were made such that there are two speeds for each direction depending on how far the esp32c3 board is tilted. Within ranges of 50 to 500, the mouse is only moved by a factor of 5, but past 500, the mouse is moved by a factor of 25.

## Appendix:

### 4.1:



**You**

I have an ESP32C3 connected with an I2C to an ICM-42670-P. Can you write a simple program to read the accelerometer using ESP-IDF without using memcpy or buffers?



**ChatGPT**

Certainly! To read the accelerometer data from an ICM-42670-P using ESP32C3 via I2C without using `memcpy` or buffers in ESP-IDF, you will need to directly handle the I2C communication. Below is a simplified example of how you can achieve this.

The following example assumes you have basic setup done for ESP-IDF, including configuring the I2C driver. This code will simply read the accelerometer values from the ICM-42670-P and print them to the console.

1. **Initialization:** Set up I2C parameters and initialize the I2C driver.
2. **Device Configuration:** Configure the ICM-42670-P to start measuring.
3. **Data Acquisition:** Continuously read the accelerometer data.

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/i2c.h"
#include "esp_system.h"
#include "esp_log.h"

#define I2C_MASTER_SCL_IO      22 // Change it according to your board configuration
#define I2C_MASTER_SDA_IO     21 // Change it according to your board configuration
#define I2C_MASTER_NUM        I2C_NUM_0
#define I2C_MASTER_FREQ_HZ    400000
#define I2C_MASTER_TX_BUF_DISABLE 0
#define I2C_MASTER_RX_BUF_DISABLE 0
#define ICM42670_ADDR          0x68 // Device address of ICM-42670-P

// Registers and values specific for ICM-42670-P
#define REG_BANK_SEL           0x76
#define REG_INT_CONFIG         0x14
#define REG_PWR_MGMT_1         0x06
#define REG_ACCEL_XOUT_H       0x1F // Starting address of accelerometer data (0x1F to 0x20)
```

```

// Function to initialize I2C
static esp_err_t i2c_master_init() {
    int i2c_master_port = I2C_MASTER_NUM;
    i2c_config_t conf = {
        .mode = I2C_MODE_MASTER,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .sda_pullup_en = GPIO_PULLUP_ENABLE,
        .scl_pullup_en = GPIO_PULLUP_ENABLE,
        .master.clk_speed = I2C_MASTER_FREQ_HZ,
    };
    i2c_param_config(i2c_master_port, &conf);
    return i2c_driver_install(i2c_master_port, conf.mode, I2C_MASTER_TX_BUF_DISABLE,
}

// Function to read bytes from a given register
static esp_err_t i2c_read_from_register(uint8_t reg, uint8_t *data, size_t len) {
    if (len == 0) return ESP_OK; // No need to read anything

    i2c_cmd_handle_t cmd = i2c_cmd_link_create();
    i2c_master_start(cmd);

```

```

    i2c_master_write_byte(cmd, (ICM42670_ADDR << 1) | I2C_MASTER_WRITE, true);
    i2c_master_write_byte(cmd, reg, true);
    i2c_master_start(cmd);
    i2c_master_write_byte(cmd, (ICM42670_ADDR << 1) | I2C_MASTER_READ, true);
    if (len > 1) {
        i2c_master_read(cmd, data, len - 1, I2C_MASTER_ACK);
    }
    i2c_master_read_byte(cmd, data + len - 1, I2C_MASTER_NACK);
    i2c_master_stop(cmd);
    esp_err_t ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 1000 / portTICK_RATE_MS);
    i2c_cmd_link_delete(cmd);
    return ret;
}

void app_main(void) {
    esp_err_t ret;
    uint8_t data[6]; // To store accelerometer data

    ret = i2c_master_init();
    if (ret != ESP_OK) {
        ESP_LOGE("I2C", "I2C initialization failed");
        return;
    }
}

```

```

while (1) {
    ret = i2c_read_from_register(REG_ACCEL_XOUT_H, data, sizeof(data));
    if (ret == ESP_OK) {
        int16_t ax = (int16_t)((data[0] << 8) | data[1]);
        int16_t ay = (int16_t)((data[2] << 8) | data[3]);
        int16_t az = (int16_t)((data[4] << 8) | data[5]);
        printf("Accelerometer values: ax=%d, ay=%d, az=%d\n", ax, ay, az);
    } else {
        ESP_LOGE("I2C", "Failed to read accelerometer data");
    }
    vTaskDelay(500 / portTICK_PERIOD_MS); // Delay 500 ms
}
}

```

## Explanation:

- **Initialization:** The I2C is initialized with GPIO 22 as SCL and GPIO 21 as SDA.
- **Reading Data:** Data from the accelerometer is read by specifying the starting register (`REG_ACCEL_XOUT_H`) and the number of bytes to read (6 bytes for X, Y, Z axes each consisting of 2 bytes).



This basic example continuously reads the accelerometer data and prints it out. Make sure to adjust the GPIO pins and I2C settings according to your specific hardware setup. Additionally, depending on your sensor's configuration, you might need to write configuration registers to start measurements or adjust settings, which isn't covered here for simplicity.