Project Overview

This project serves to create a lightweight, simple library for the MPU6050 inertial measurement unit (IMU). The MPU6050 is a cheap 6 axis sensor, combining a 3 axis accelerometer with a 3-axis gyroscope. Due to it's low cost, power draw and small size, it's ideal for consumer devices. One such implementation could be on a tracking device which can be left in a low power mode until movement is detected. It uses the I2C communication protocol to communicate with embedded devices. The library I wrote is intended for use on STM32 microprocessors and uses ST's hardware libraries to handle I2C communications with the chip. It's intended to be as simple as possible to keep development time low for applications where more complex functions aren't required.

This project used an MPU6050 IMU on a readily available GY-521 breakout board. Likewise, the STM32f303re microprocessor came on a Nucleo-branded breakout board from ST microelectronics. All connections are made using junction cables as shown below.

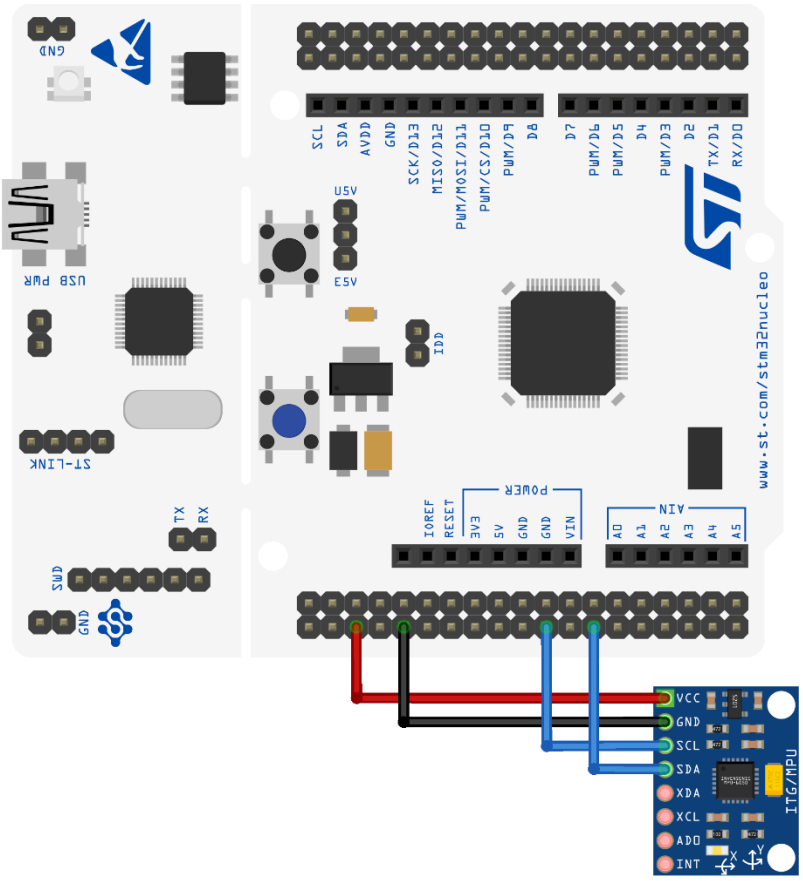

Figure 1: Hardware Schematic

Implementation

My library utilizes ST Microelectronics' hardware abstraction layer to interface with the MPU6050 over I2C. In particular, it makes use of the HAL_I2C_Mem_Read() function, which combines I2C write and read functions into a single function. My code uses it to transmit the device and required memory register addresses to the MPU6050, and reads it into a buffer. The function my library uses to read from the gyroscope data registers is shown below.

```
HAL_StatusTypeDef MPU6050_getGyro(I2C_HandleTypeDef *hi2c, uint8_t MPU6050_ADDR, float *buf) {
    uint8_t gyro_raw[6]; //Probably a more efficient way to allocate data

    HAL_StatusTypeDef status; //Status handle

    status = HAL_I2C_Mem_Read(hi2c,MPU6050_ADDR,MPU6050_GYRO_XOUT_H,I2C_MEMADD_SIZE_8BIT,gyro_raw,6, HAL_MAX_DELAY

    buf[0] = (int16_t)(gyro_raw[0] << 8 | gyro_raw[1]); //Separate raw data for each axis into buffer
    buf[1] = (int16_t)(gyro_raw[2] << 8 | gyro_raw[3]);
    buf[2] = (int16_t)(gyro_raw[4] << 8 | gyro_raw[5]);

    buf[0] = buf[0] / 131.0f; //Divide by scaling factors to deg/s
    buf[1] = buf[1] / 131.0f;
    buf[2] = buf[2] / 131.0f;

    return status;
}
```

It takes an I2C handle, relevant addresses, and a pointer to a data buffer as inputs. The gyroscope data is written into the buffer passed as an input, and the function returns a status message which can be used for verification.

Verification

I utilized the status returned by the HAL I2C functions to verify that each function call returned good data. My library simply returns this status message so that the user can utilize it. My test script will not enter the main loop while the MPU6050_init function returns a negative status message. Similarly in the main loop, acceleration data is not updated when the MPU6050_getAccel function call returns a status other than *HAL_OK*, which signifies a successful read/write.

Numerical Methods

For my test script, I implemented a *very* simple time integration program to estimate position from the MPU6050's acceleration data. This assumes that the sensor does not rotate while it moves, and that the +Z axis is perfectly aligned with the gravity vector. This does demonstrate that the sensor functions as intended, however it is quickly overwhelmed with noise. The HAL_getTick() function provided a reliable way to get the time between sensor readings, using an onboard timer. Resulting data can be seen below:

Expression	Type	Value	Address
accel_buf	float [3]	[3]	0x20000104
accel_buf[0]	float	-0.10546875	0x20000104
accel_buf[1]	float	0.953613281	0x20000108
accel_buf[2]	float	0.219238281	0x2000010c
x_vel	float	-0.00352067873	0x2000011c
y_vel	float	0.0279786177	0x20000120
z_vel	float	-0.0224402025	0x20000124
x_pos	float	-1.05620366e-005	0x20000128
y_pos	float	8.39358545e-005	0x2000012c
z_pos	float	-6.73206087e-005	0x20000130
Add new expression			

I2C Debugging

I had a lot of trouble getting I2C to work early on. I used the following script to ensure that the I2C addresses were correct, and a device was connected. Through this, I discovered the i2c1 pins were configured to alternate GPIO.

```
for(int i=1; i<128; i++)
    {
        ret = HAL_I2C_IsDeviceReady(&hi2c1, (uint16_t)(i<<1), 3, 5);
        if (ret != HAL_OK) /* No ACK Received At That Address */
        {

        }
        else if(ret == HAL_OK)
        {
          address = (uint16_t)i;

        }
    }
```

With the GPIO correctly configured, this returned HAL_OK for address = 0x68, as per the chip datasheet

Another issue I encountered was that the MPU6050 requires a register to be written to activate the device. This must be done before reading sensor data, otherwise data registers will return 0x00. The function I wrote to initialize the sensor is below:

```c
HAL_StatusTypeDef MPU6050_init(I2C_HandleTypeDef *hi2c, uint8_t MPU6050_ADDR) {

    uint8_t data = 0x00; //0x00 to clear sleep bit
    HAL_StatusTypeDef status;

    status = HAL_I2C_Mem_Write(hi2c,MPU6050_ADDR,0x6B,I2C_MEMADD_SIZE_8BIT,&data,1,HAL_MAX_DELAY);

    return status;
}
```

Next Steps

This is a very simple I2C driver, and there's a lot which can be improved with it.

- Configuration functions: Add functions to modify the settings on the MPU6050, especially the acceleration sensitivity to enable higher-g operation
- A full 6dof position estimation script, using both acceleration and gyroscope data to estimate position over time.
- Sleep/wake functions to limit power draw

Overall, this was a great learning experience for working with I2C and embedded development. Working with I2C at the hardware level gave me a good understanding of how it works and where it's imitations lie.

Progress Log

Date	Work Completed
12/10	Initial hardware setup, first "hello world" program on the STM32 nucleo board
12/12	Learned how to use HAL i2c functions with a simpler HTU2X sensor. Began working with the MPU6050
12/14	Wrote functions for accessing MPU6050 accel/gyro/temp data registers
12/15	Test script, documentation
12/16	Readme.md