CPE301 – SPRING 2019

Design Assignment 6

Student Name: Do Le

Student #: 2001183621

Student Email: led2@unlv.nevada.edu

Primary Github address: <https://github.com/DoVietLe>

Directory: <https://github.com/DoVietLe/assignments>

The goal of the assignment is to develop the above code to do the following:

1. Interface the provided MPU-6050 6-DOF IMU Sensor to the ATmega328p using the I2C interface. Using the earlier developed code for UART, display the accelerometer and gyro data to the UART Terminal. Visualizing the accelerometer and gyro values using the serial plotter.

2. Apply complementary filtering\* on the sensor data, calculate the smooth/filtered pitch and roll angles and display the filtered values in the graph.

1. **COMPONENTS LIST AND CONNECTION BLOCK DIAGRAM w/ PINS**

Components

• XPlained Mini Board

* ATmega328PB
* Programmer

• MPU-6050

• Arduino Custom Shield

• Atmel Studios

* Compiler
* Data Visualizer
* Serial Monitor

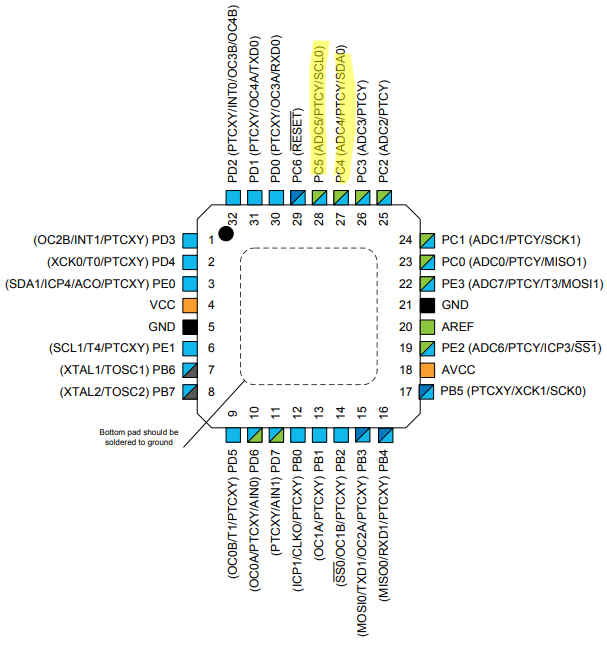
Block Diagram

Figure 1: Pinout

PC5 used for SCL & PC4 used for SDA for I2C.

1. **INITIAL/MODIFIED/DEVELOPED CODE OF TASK 1/A**

Task 1 Code

#define *F\_CPU* 16000000UL

#include <avr/io.h>

#include <util/delay.h>

#define BIT\_RATE\_VAL 3

#define SLAVE\_ADDRESS 0xD0

#define DIVIDER\_ADDRESS 0x19

#define POWER\_ADDRESS 0x6B

#define CONFIG\_ADDRESS 0x1A

#define GYRO\_CONFIG\_ADDRESS 0x1B

#define INTERRUPT\_ADDRESS 0x38

#define ACC\_START\_ADDRESS 0x3B

#define GYRO\_START\_ADDRESS 0x43

#define INTERRUPT\_ADDRESS 0x38

#define ACC\_SCALE 16384.0

#define GYRO\_SCALE 16.4

#define UBBR\_VALUE 103

volatile char myIntString[200];

// I2C functions.

void i2c\_init() {

// Initializes settings.

TWBR0 = BIT\_RATE\_VAL; // Sets bit rate.

TWSR0 |= (0<<TWPS1)|(0<<TWPS0); // Sets pre-scaler value to 1.

}

void i2c\_start() {

// Generates a start condition on SDA line.

TWCR0 |= (1<<TWSTA)|(1<<TWEN)|(1<<TWINT); // Generates start condition.

while ( !(TWCR0&(1<<TWINT)) ); // Waits for TWINT to go high.

}

void i2c\_start\_wait(*uint8\_t* slaveAddress)

{

*uint8\_t* status = 0;

// Repeatedly attempts to send start and address until an ACK is received.

while (1) {

TWCR0 = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT); // Generates start condition.

while (!(TWCR0 & (1<<TWINT)));

status = TWSR0 & 0xF8; // Checks status for ACK.

if (status != 0x08)

continue; // Sends slave address.

TWDR0 = slaveAddress;

TWCR0 = (1<<TWEN)|(1<<TWINT);

while (!(TWCR0 & (1<<TWINT)));

status = TWSR0 & 0xF8; // Checks status for ACK.

if (status != 0x18 )

{

i2c\_stop(); // Stop if conditions aren't met.

continue;

}

break;

}

}

void i2c\_stop() {

// Generates a stop condition on the SDA line.

TWCR0 |= (1<<TWSTO)|(1<<TWEN)|(1<<TWINT); // Generates stop condition.

while ( (TWCR0&(1<<TWSTO)) ); // Waits for TWINT to go high.

}

void i2c\_send(*uint8\_t* data) {

// Sends data through the SDA line.

TWDR0 = data;

TWCR0 |= (1<<TWEN)|(1<<TWINT);

while ( !(TWCR0&(1<<TWINT)) ); // Waits for TWINT to go high.

}

char i2c\_read(*uint8\_t* ack) {

TWCR0 = (1<<TWEN)|(1<<TWINT)|(ack ? (1<<TWEA) : 0x00);

while ( !(TWCR0&(1<<TWINT)) ); // Waits for TWINT to go high.

return TWDR0;

}

// MP6050 functions

void initializeMP6050() {

// Configures divider pre-scaler.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(DIVIDER\_ADDRESS); // Sends first register address.

i2c\_send(0x07); // Sample rate register value as 7. (Sample at 1kHz)

i2c\_stop(); // Generates stop condition.

// Configures power.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(POWER\_ADDRESS); // Sends register address.

i2c\_send(0x01); // PLL with x-axis reference.

i2c\_stop(); // Generates stop condition.

// Configures configuration register.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(CONFIG\_ADDRESS); // Sends register address.

i2c\_send(0x00); // Sets Fs as 8kHz.

i2c\_stop(); // Generates stop condition.

// Configures power.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(GYRO\_CONFIG\_ADDRESS); // Sends register address.

i2c\_send(0x18); // Sets full-scale range as +/- 2000 degrees.

i2c\_stop(); // Generates stop condition.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(INTERRUPT\_ADDRESS); // Sends register address.

i2c\_send(0x01); // Enables DATA\_RDY\_EN.

i2c\_stop(); // Generates stop condition.

}

void readRawData(float accData[], float gyroData[]) {

// Sets pointer.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(ACC\_START\_ADDRESS); // Sends XH address.

// Starts reading in accelerometer data.

i2c\_start(); // Sends another start condition.

i2c\_send(SLAVE\_ADDRESS|0x01); // Sends slave address in read mode.

accData[0] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(1) );

accData[1] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(1) );

accData[2] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(0) );

i2c\_stop(); // Generates stop condition.

// Sets pointer.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(GYRO\_START\_ADDRESS); // Sends gyro XH address.

// Starts reading in gyro data.

i2c\_start(); // Sends another start condition.

i2c\_send(SLAVE\_ADDRESS|0x01); // Sends slave address in read mode.

gyroData[0] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(1) );

gyroData[1] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(1) );

gyroData[2] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(0) );

i2c\_stop(); // Generates stop condition.

}

void scaleData(float accData[], float gyroData[]) {

for (*uint8\_t* i = 0; i < 3; i++) {

accData[i] /= ACC\_SCALE;

gyroData[i] /= GYRO\_SCALE;

}

}

// USART functions.

// Enables the UART and it's transmitter.

void enableTransmit() {

// Sets port.

DDRD |= 0x02;

// Sets baud rate.

UBRR0 = UBBR\_VALUE;

// Sets UART settings.

UCSR0C = 0b00000110; // to async mode with no parity

// 2 stop bits, and a frame of 8 bits.

UCSR0B = 0b00001000; // Enables transmit line.

}

// Disables the transmitter on UART.

void disableTransmit() {

UCSR0B &= ~(1<<TXEN0);

}

// Sends one byte of data out the TX line.

void sendByte(*uint8\_t* b) {

while (!(UCSR0A & (1<<UDRE0))); // Waits until the UDR0 register is ready.

UDR0 = b; // Sends the byte.

}

void sendString(char string[]) {

int i = 0;

while (string[i] != '\0')

sendByte(string[i++]);

}

int main() {

// Variables.

float accData[3], gyroData[3];

// Initialize modules.

i2c\_init();

enableTransmit();

*\_delay\_ms*(150); // Wait some time.

initializeMP6050();

while (1) {

*\_delay\_ms*(1000);

readRawData(accData, gyroData); // Read data from MP6050.

scaleData(accData, gyroData); // Applies scale value.

*snprintf*(myIntString, 200, "A = <%f, %f, %f> G = <%f, %f, %f> \n\r",

accData[0], accData[1], accData[2],

gyroData[0], gyroData[1], gyroData[2]); // Converts to a string float.

sendString(myIntString); // Sends data through USART.

}

}

1. **DEVELOPED MODIFIED CODE OF TASK 2/A from TASK 1/A**

Task 2 Code

#define *F\_CPU* 16000000UL

#include <avr/interrupt.h>

#include <avr/io.h>

#include <util/delay.h>

#define BIT\_RATE\_VAL 3

#define SLAVE\_ADDRESS 0xD0

#define DIVIDER\_ADDRESS 0x19

#define POWER\_ADDRESS 0x6B

#define CONFIG\_ADDRESS 0x1A

#define GYRO\_CONFIG\_ADDRESS 0x1B

#define INTERRUPT\_ADDRESS 0x38

#define ACC\_START\_ADDRESS 0x3B

#define GYRO\_START\_ADDRESS 0x43

#define INTERRUPT\_ADDRESS 0x38

#define ACC\_SCALE 16384.0

#define GYRO\_SCALE 16.4

#define UBBR\_VALUE 103

#define TIMER\_10MILLI 624

#define dt 0.01

volatile char myIntString[70];

volatile float accData[3], gyroData[3];

volatile float pitch = 0;

volatile float roll = 0;

// I2C functions.

void i2c\_init() {

// Initializes settings.

TWBR0 = BIT\_RATE\_VAL; // Sets bit rate.

TWSR0 |= (0<<TWPS1)|(0<<TWPS0); // Sets pre-scaler value to 1.

}

void i2c\_start() {

// Generates a start condition on SDA line.

TWCR0 |= (1<<TWSTA)|(1<<TWEN)|(1<<TWINT); // Generates start condition.

while ( !(TWCR0&(1<<TWINT)) ); // Waits for TWINT to go high.

}

void i2c\_start\_wait(*uint8\_t* slaveAddress)

{

*uint8\_t* status = 0;

// Repeatedly attempts to send start and address until an ACK is received.

while (1) {

TWCR0 = (1<<TWSTA)|(1<<TWEN)|(1<<TWINT); // Generates start condition.

while (!(TWCR0 & (1<<TWINT)));

status = TWSR0 & 0xF8; // Checks status for ACK.

if (status != 0x08)

continue; // Sends slave address.

TWDR0 = slaveAddress;

TWCR0 = (1<<TWEN)|(1<<TWINT);

while (!(TWCR0 & (1<<TWINT)));

status = TWSR0 & 0xF8; // Checks status for ACK.

if (status != 0x18 )

{

i2c\_stop(); // Stop if conditions aren't met.

continue;

}

break;

}

}

void i2c\_stop() {

// Generates a stop condition on the SDA line.

TWCR0 |= (1<<TWSTO)|(1<<TWEN)|(1<<TWINT); // Generates stop condition.

while ( (TWCR0&(1<<TWSTO)) ); // Waits for TWINT to go high.

}

void i2c\_send(*uint8\_t* data) {

// Sends data through the SDA line.

TWDR0 = data;

TWCR0 |= (1<<TWEN)|(1<<TWINT);

while ( !(TWCR0&(1<<TWINT)) ); // Waits for TWINT to go high.

}

char i2c\_read(*uint8\_t* ack) {

TWCR0 = (1<<TWEN)|(1<<TWINT)|(ack ? (1<<TWEA) : 0x00);

while ( !(TWCR0&(1<<TWINT)) ); // Waits for TWINT to go high.

return TWDR0;

}

// MP6050 functions

void initializeMP6050() {

// Configures divider pre-scaler.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(DIVIDER\_ADDRESS); // Sends first register address.

i2c\_send(0x07); // Sample rate register value as 7. (Sample at 1kHz)

i2c\_stop(); // Generates stop condition.

// Configures power.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(POWER\_ADDRESS); // Sends register address.

i2c\_send(0x01); // PLL with x-axis reference.

i2c\_stop(); // Generates stop condition.

// Configures configuration register.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(CONFIG\_ADDRESS); // Sends register address.

i2c\_send(0x00); // Sets Fs as 8kHz.

i2c\_stop(); // Generates stop condition.

// Configures power.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(GYRO\_CONFIG\_ADDRESS); // Sends register address.

i2c\_send(0x18); // Sets full-scale range as +/- 2000 degrees.

i2c\_stop(); // Generates stop condition.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(INTERRUPT\_ADDRESS); // Sends register address.

i2c\_send(0x01); // Enables DATA\_RDY\_EN.

i2c\_stop(); // Generates stop condition.

}

void readRawData(float accData[], float gyroData[]) {

// Sets pointer.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(ACC\_START\_ADDRESS); // Sends XH address.

// Starts reading in accelerometer data.

i2c\_start(); // Sends another start condition.

i2c\_send(SLAVE\_ADDRESS|0x01); // Sends slave address in read mode.

accData[0] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(1) );

accData[1] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(1) );

accData[2] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(0) );

i2c\_stop(); // Generates stop condition.

// Sets pointer.

i2c\_start\_wait(SLAVE\_ADDRESS); // Generates start condition.

i2c\_send(GYRO\_START\_ADDRESS); // Sends gyro XH address.

// Starts reading in gyro data.

i2c\_start(); // Sends another start condition.

i2c\_send(SLAVE\_ADDRESS|0x01); // Sends slave address in read mode.

gyroData[0] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(1) );

gyroData[1] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(1) );

gyroData[2] = ( ((int)i2c\_read(1) << 8) + (int)i2c\_read(0) );

i2c\_stop(); // Generates stop condition.

}

void scaleData(float accData[], float gyroData[]) {

for (*uint8\_t* i = 0; i < 3; i++) {

accData[i] /= ACC\_SCALE;

gyroData[i] /= GYRO\_SCALE;

}

}

// USART functions.

// Enables the UART and it's transmitter.

void enableTransmit() {

// Sets port.

DDRD |= 0x02;

// Sets baud rate.

UBRR0 = UBBR\_VALUE;

// Sets UART settings.

UCSR0C = 0b00000110; // to async mode with no parity

// 2 stop bits, and a frame of 8 bits.

UCSR0B = 0b00001000; // Enables transmit line.

}

// Disables the transmitter on UART.

void disableTransmit() {

UCSR0B &= ~(1<<TXEN0);

}

// Sends one byte of data out the TX line.

void sendByte(*uint8\_t* b) {

while (!(UCSR0A & (1<<UDRE0))); // Waits until the UDR0 register is ready.

UDR0 = b; // Sends the byte.

}

void sendString(char string[]) {

int i = 0;

while (string[i] != '\0')

sendByte(string[i++]);

}

// Timer functions.

void initializeTimer() {

// Sets up TOP.

OCR1A = TIMER\_10MILLI;

// Sets timer.

TCCR1A = 0x00; // Sets timer to CTC mode with OCR1A TOP.

TCCR1B = (1<<WGM12)|(1<<CS12); // Sets pre-scaler to 256.

// Sets up interrupt.

TIMSK1 = (1<<OCIE1A); // Enables OCR1A compare match interrupt.

TIFR1 |= (1<<OCF1A); // Clears interrupt flag.

TCNT1 = 0;

}

int main() {

// Initialize modules.

i2c\_init();

enableTransmit();

*\_delay\_ms*(150); // Wait some time.

initializeMP6050();

initializeTimer();

sei();

while (1) {

readRawData(accData, gyroData); // Read data from MP6050.

//scaleData(accData, gyroData); // Applies scale value.

*snprintf*(myIntString, 70, "Pitch: %f, Roll: %f\n", pitch, roll);

sendString(myIntString);

//snprintf(myIntString, 70, "%f, %f, %f, %f, %f, %f\n",

//accData[0], accData[1], accData[2],

//gyroData[0], gyroData[1], gyroData[2]); // Converts to a string float.

//sendString(myIntString); // Sends data through USART.

}

}

ISR(TIMER1\_COMPA\_vect) {

float pitchAcc, rollAcc;

// Integrate the gyroscope data -> int(angularSpeed) = angle

pitch += ((float)gyroData[0] / GYRO\_SCALE) \* dt;

// Angle around the X-axis

roll -= ((float)gyroData[1] / GYRO\_SCALE) \* dt;

// Angle around the Y-axis

// Compensate for drift with accelerometer data if !bullshit

// Sensitivity = -2 to 2 G at 16Bit -> 2G = 32768 && 0.5G = 8192

int forceMagnitudeApprox = *abs*(accData[0]) + *abs*(accData[1]) + *abs*(accData[2]);

if (forceMagnitudeApprox > 8192 && forceMagnitudeApprox < 32768)

{

// Turning around the X axis results in a vector on the Y-axis

pitchAcc = *atan2f*((float)accData[1], (float)accData[2]) \* 180 / *M\_PI*;

pitch = pitch \* 0.98 + pitchAcc \* 0.02;

// Turning around the Y axis results in a vector on the X-axis

rollAcc = *atan2f*((float)accData[0], (float)accData[2]) \* 180 / *M\_PI*;

roll = roll \* 0.98 + rollAcc \* 0.02;

}

TIFR1 |= (1<<OCF1A); // Clears interrupt flag.

}

1. **SCHEMATICS**

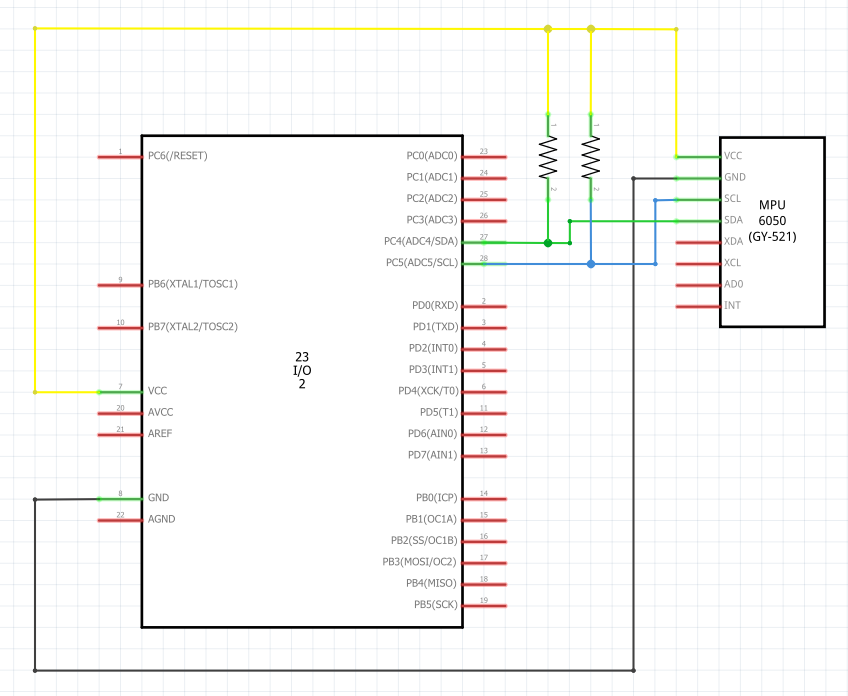


Figure 2: Schematic

1. **SCREENSHOTS OF EACH TASK OUTPUT (ATMEL STUDIO OUTPUT)**
2. **SCREENSHOT OF EACH DEMO (BOARD SETUP)**

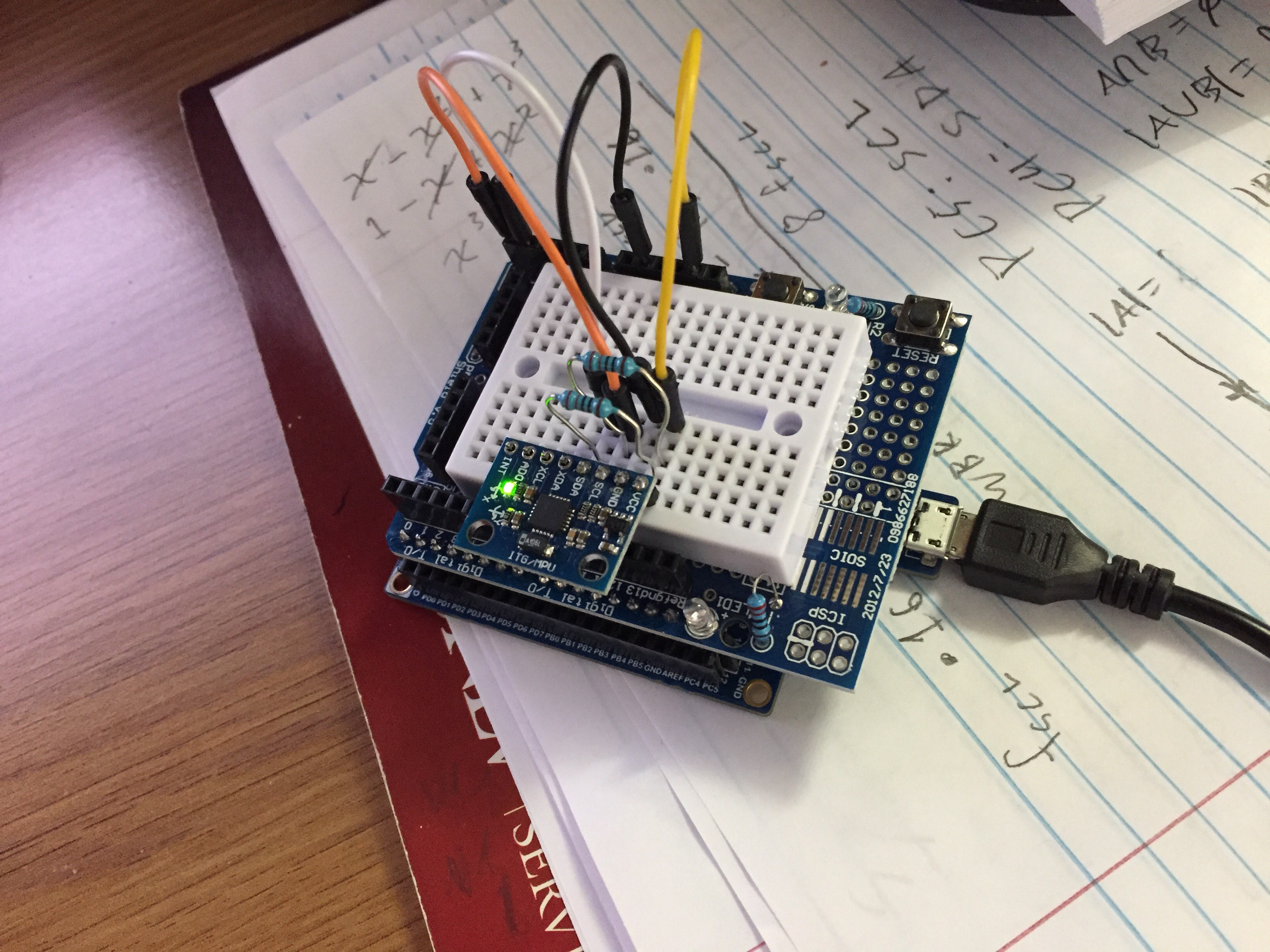


Figure 3: Board setup

1. **VIDEO LINKS OF EACH DEMO**

Task 1 Video

https://www.youtube.com/watch?v=\_Bdgh7xb5s8

Task 2 Video

https://www.youtube.com/watch?v=i6E5AH5OmBI

1. **GITHUB LINK OF THIS DA**

https://github.com/DoVietLe/assignments/tree/master/ESD301/LAB06

**Student Academic Misconduct Policy**

<http://studentconduct.unlv.edu/misconduct/policy.html>

“This assignment submission is my own, original work”.

Do Viet Le