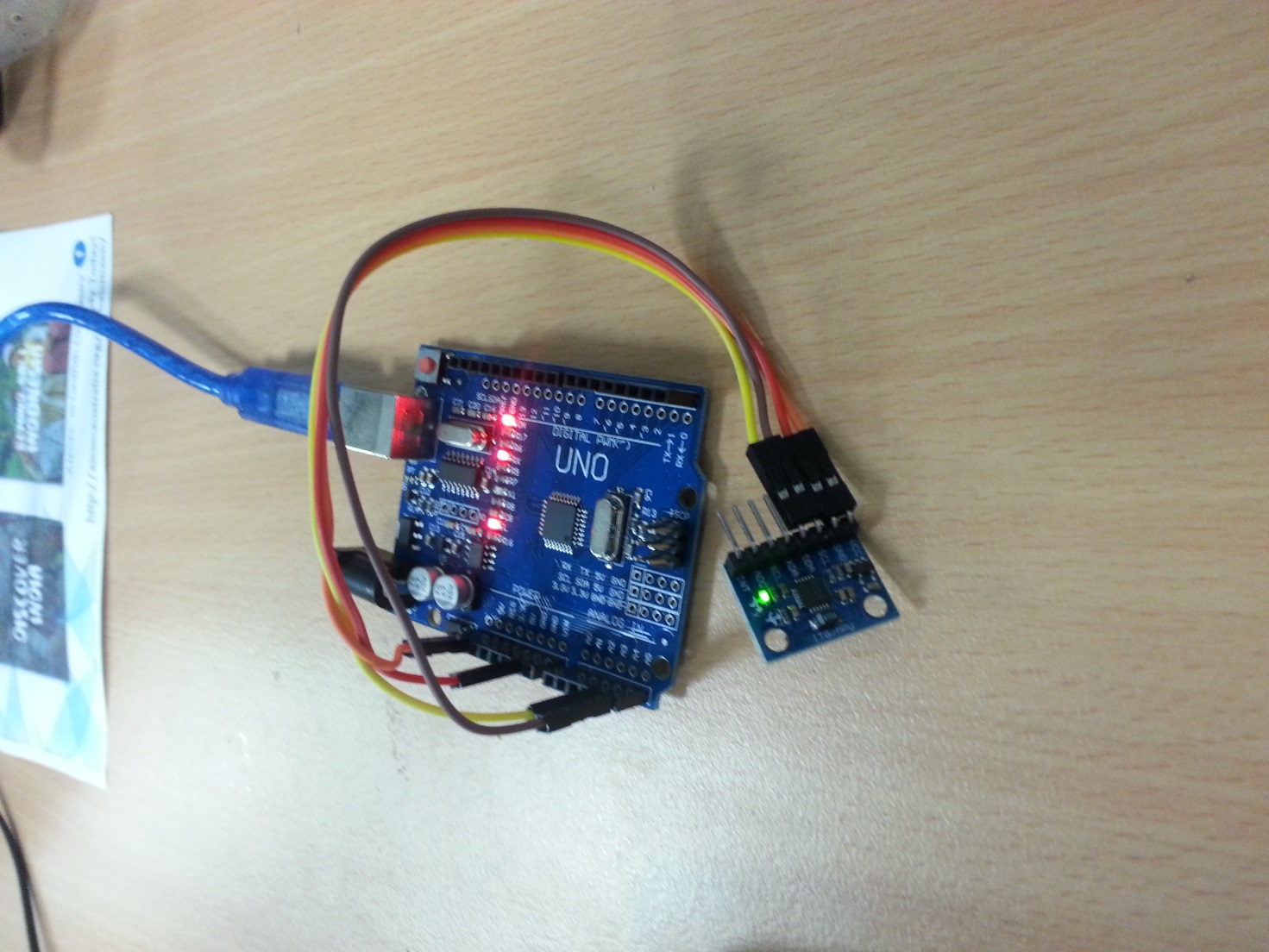
# Lab 5: IMU + Pointers

In this lab you will be using the IMU unit to read some values including a 3D vector representing the direction the IMU is facing as well as a 3D vector representing the current acceleration of the IMU, you will also be recording the temperature using the IMU. You will practice using pointers by creating a function which conveniently fills in variables passed in as pointers to these values.

Steps:

1. Read the tutorial on using the IMU
2. Do the lab included at the end of this document

## Tutorial: IMU: Internal Measurement Unit

In this tutorial we will drive the IMU. The IMU provides the Arduino with acceleration and gyroscopic information as well as temperature information. This is provided in the form of two vectors \_acceleration and \_direction.The wiring is shown below, but the wiring is identical to the compass tutorial: vcc to vcc, gnd to gnd, SLC to A5 and SDA to A4.

In this tutorial, we will define our own Vector struct and use the Wire library module. Here are our includes, global variables and our Vector struct called Vec3:

#include<Wire.h>

const int MPU\_addr=0x68; // I2C address of the MPU-6050

struct Vec3{

int x, y, z;

void init(int x, int y, int z){ this->x = x, this->y = y, this->z = z; }

String toString(){ return String(x) + ", " + String(y) + ", " + String(z); }

};

We use an address id of 0x68 when communicating with the IMU. We also will be using a basic struct called Vec3, it has 3 member variables, x, y and z. It also has an init(x,y,z) function for setting these variables and a toString function which returns a string representation of the vector.

In our setup function, we initialize communication with the IMU by writing the address idwhich we are communicating through (0x68), before waking up the IMU by writing 0x6B and 0x00. We also setup our Serial object baud rate. We have also included a function called read16bits, which uses the Wire module to do just that. It reads one byte, before shifting those bits over to the left 8 times, and does a bit wise or with the second byte it reads. It returns these 16 bits back as our standard 32 bit integer.

void setup(){

Wire.begin();

Wire.beginTransmission(MPU\_addr);

Wire.write(0x6B); // PWR\_MGMT\_1 register

Wire.write(0); // set to zero (wakes up the MPU-6050)

Wire.endTransmission(true);

Serial.begin(9600);

}

int read16Bits(){

return Wire.read()<<8|Wire.read();

}

In our loop function (shown below) we continue transmission to the IMU by writing the destination address id once again, then requesting to read our vectors and temperature from it’s registers.

Next we read the acceleration vector, the temperature and the direction (gyroscope) vector using our read16Bits function, each component of a vector is 16 bits and the temperature is just 16 bits.

So we declare our vectors and temperature variables as Vec3s and floats. And use the read16Bits function, passing the result to our init functions with the Vec3s or straight to our \_temperature variable (since it is a singular value). Finally we use the toString functions in the Vec3s to print our vectors to the serial monitor.

void loop(){

Wire.beginTransmission(MPU\_addr);

Wire.write(0x3B); // starting with register 0x3B (ACCEL\_XOUT\_H)

Wire.endTransmission(false);

Wire.requestFrom(MPU\_addr,14,true); // request a total of 14 registers

//initialize our vectors and temperature

Vec3 \_acceleration, \_direction;

float \_temperature;

\_acceleration.init(read16Bits(), read16Bits(), read16Bits());

\_temperature = ((float)read16Bits())/340.00+36.53;

\_direction.init(read16Bits(), read16Bits(), read16Bits());

Serial.print("direction: " + \_direction.toString());

Serial.print(",\tacceleration: " + \_acceleration.toString());

Serial.println(",\ttemperature: " + String(\_temperature));

delay(300);

}

### Lab 5: IMU and Pointers

In this lab, you will be required to write a function which hides away the complexity of retrieving the values generated by the IMU. Since the return values of this function will be 6 integers and a float, a single return value will not suffice. We must use pointers. Write a function to retrieve the IMU values with the following signature:

void retrieveIMUData(int \* xDirection, int \* yDirection, int \* zDirection,

int \* xAccel, int \* yAccel, int \* zAccel, float \* temperature);

Retrieve the values and store them in the locations pointed to by the pointers given as input to the function.