# Compass Read and Publish

## GOAL:

* Read HMC5883L compass sensor with Atmega328P over I2C.
  + read x axis magnitude in milligauss
  + read y axis magnitude in milligauss
  + read z axis magnitude in milligauss
  + calculate direction sing pythagoean theorem in R3.
* Send data to ESP8266 over UART.
* Publish data on api.ThingSpeak.com

## DELIVERABLES:

The final deliverable is a device that will capture magnetic field magnitude data on three axis x, y, and z. Also, publish data to a webserver for viewing on a ThingSpeak page.

## LITERATURE SURVEY:

Mobile devices and vehicles such as robots often need to be aware of their orientation. One way to measure it, is by measuring the magnetic field acting on the robot [1]. The HMC5883L is one sensor that can accomplish this task. It is often also necessary to gather data of such sensors and publish it for other other services or entities to read and process. For this task, the ESP8266 Wi-Fi module will be used.

All devices will be running at 5 V with the Wi-Fi module consuming the most power drawing up to 250 mA. This means the system can be powered by a small.

## COMPONENTS:

### Hardware:

* **Atmega328P** is an 8-bit AVR microcontroller (MCU). For this project, it is used for data collection, and preparing it for output. It simply configures the HMC5883L and outputs a formatted string that represents the read data over UART0 at a baudrate of 9600.

The MCU communicates with the HMC5883L over I2C. At start up, the MCU configures the HMC5883L with the configuration explained in the following section. The MCU reads the x, y, and z registers of the HMC5883L, which are interpreted as magnetic field vectors on each axis. Then, the MCU calculates the direction on the plane that is parallel to the PCB or breadboard it is mounted on. For the demo of this project, the xz-plane is parallel to the breadboard. The following formula is used to calculate the direction of the magnetic field with respect to the breadboard:

Once the data is retrieved, the MCU outputs x, y, z, and direction values to UART0 with the following format:

<x\_value>\t< y\_value >\t< z\_value>\t<direction\_deg>

with each value delimited by tabs, followed by a linefeed ‘\n’. Then the process restarts from data retrieval.

* **HMC5883L** is the magnetometer sensor used, also known as a compass. This compass reads and outputs values in units of gauss. It uses a 12-bit ADC which means that the output values have the range of a 12-bit 2’s complement number (-2048 to 2047) over two 8-bit registers. Depending on the configuration of this sensor, the output will be within certain ranges. The range can be changed if the output is expected to overflow. The configuration used for this project is Ga. For example, if the range of the magnetic field in the environment of the sensor is outside the range of Ga, then choose a higher range configuration.

Since the output range is -2048 to 2047 regardless of magnetic field range, the output values must be mapped to the magnetic field range with

Where *value* would be the value in units of Ga. *rv* is the read value from the sensor*. rmin* and *rmax* are the output range limits. -2048 to 2048 in every case. *out­min* and *outmax* are the configured range limits. In this case, -1.3 and 1.3 respectively. This formula will map a value from -2048 to 2048 to a value in the range of -1.3 to 1.3.

* **ESP8266** is the wi-fi module used in this project. The task of this device is to transmit data it receives over UART, and publish it on a webserver. The two firmware explored in this project were NodeMCU and AT Firmware. This wi-fi module is also programmable with a regular Arduino IDE with regular Arduino pin assignments.

If AT Firmware is used, a connected device must control the ESP8266 by sending it AT commands over UART. Care must be taken when a command fails. With failed commands, the wi-fi module will respond with appropriate failure message such as “ERROR.” To initiate the module as a wi-fi station, it must be enabled as such, and then connect to a known network. Each command sent must be terminated by a return ‘\r’ and a linefeed ‘\n’.

AT+CWMODE=1 to enable as a wi-fi station. Set to 2 to enable it as access point, or 3 for both. A successful command will return “OK”.

AT+CWJAP=”SSID”,”PASSWORD” to connect to a wi-fi access point. Replace “SSID” and “PASSWORD” accordingly. A successful command will return “WIFI GOT IP” once connected and IP has been assigned.

Optionally, run AT+CIPMUX=1 to enable multiple server connections. Four connections allowed.

AT+CIPSTART="TCP","api.thingspeak.com",80 for this project to connect to thinkspeak server. Insert an integer 1-4 after the equal sign if using multiple connections. A successful command will return “OK”.

AT+CIPSEND=45 to prepare to send 45 bytes. Change the 45 accordingly. The wi-fi module will respond with a “>>” prompt. The 45 bytes of data can then be sent. The string must be appended with “\r\n\r\n”, which are part of the byte count of the string. The input is not echoed. The following string can be sent to update a field in thingSpeak.com.

GET /update?key=<api key>&field1=60 followed by “\r\n\r\n”. <api key> should be replaced with a valid API key with write permissions. The wi-fi device should respond with “SEND OK” if the data was successfully sent.

Run AT+CIPCLOSE in case the remote server does not close the connection itself.

NodeMCU can also be used to perform the same operations by sending the same string to the remote server. Code for this procedure can be found in later sections. The script running on this module listens to UART input expecting a certain formatted string. Once it is received, it is parsed, and data is stored. A timed procedure continuously posts data every 15 seconds.

* **A Breadboard** is used to connect all the hardware.
* **FTDI adapter** is used to communicate and monitor over UART with NodeMCU.

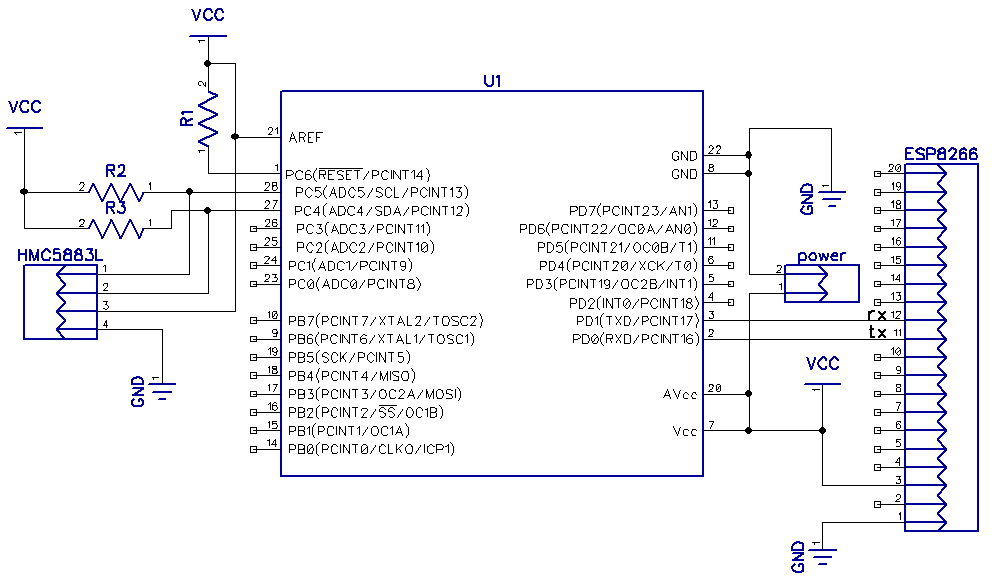
Protocols:

* **I2C** between MCU and HMC5883L.
* **UART** between MCU, ESP8266, and monitoring device (PC).
* **SPI (for programming).**

### Web service

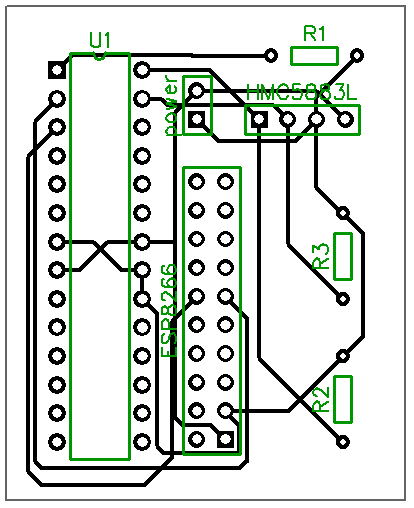
**ThingSpeak** is a free service that promotes IoT technology development.

## SCHEMATICS:



## PCB:

**All layers:**



IMPLEMENTATION:

This project has implementations in robotics and mobile devices.

* In robotics, bodies that need to be aware of their orientation can use a magnetometer to measure the orientation of the body of the robot with respect to the magnetic field acting on the sensor.   
  For example, a drone can use the HMC5883 so that it can be aware the direction it is facing. Sensor data can also be used to maintain stability on the drone’s yaw motions.
* In mobile devices such as smart phones, the sensor must be used in a more dynamic way in conjunction with other sensor data such as an accelerometer and gyroscope. One example is for GPS tracking and orientation. GPS can often provide orientation data. However, when that is not available, a magnetometer can be used to provide a display to the user its direction.  
  Since a mobile device is usually not held orthogonally to the Earth, other sensor data must be used to determine the orientation of the magnetometer’s axis to determine a plane that is tangent to the earth.

SNAPSHOTS/SCREENSHOTS:

Show snapshots/video of component implementation.

Show snapshots/video of demo (IOT/BLE/VISUALIZATION).

CODE: (with comments)

### main.c

#define F\_CPU 8000000UL // set CPU frequency

#define BAUDRATE 9600 // set baudrate

#include <avr/io.h>

#include <util/delay.h> // delay subroutines

#include <stdio.h> // standard input output

#include <stdlib.h> // standard API

#include <math.h> // for arctan2

#include "i2c.h"

#include "HMC5883L.h" // For compass module

#include "uart.h"

double calcDirection(double, double);

double map(double, double, double, double, double);

//set stream pointer

*FILE* usart0\_str = *FDEV\_SETUP\_STREAM*(USART0\_sendChar, USART0\_ReceiveChar, *\_FDEV\_SETUP\_RW*);

int main(void)

{

*int16\_t* values[3]; // array for holding axes values.

double direction; // direction of sum of vectors on a plane.

// Reassign standard input/output

*stdin* = *stdout* = &usart0\_str;

USART0\_init(); // Initiate USART0.

i2c\_init(); // Initiate I2C.

HMC5883L\_init(); // Initiate compass.

// Debugging print out.

*printf*("Program start.\n");

// Main program loop.

while (1)

{

// call procedure to read compass values on all axis.

readCompass(values);

// Get direction on the xz-plane

direction = *atan2*(values[2], values[0]) \* 180 / *M\_PI*;

// Print compass values on USART0.

*printf*("%d\t%d\t%d\t%d\n", values[0], values[1], values[2], (int)direction);

// Delay since thingspeak will not accept data faster 4 times per minute.

// (every 15 seconds.

*\_delay\_ms*(1000);

}

}

double map(double x, double in\_min, double in\_max, double out\_min, double out\_max)

// modified from source: http://www.avrfreaks.net/forum/mapping-numbers-avr

// by singelen

{

return (x - in\_min) \* (out\_max - out\_min) / (in\_max - in\_min) + out\_min;

}

### uart.h

Most of the code in this file is based on class provided material.

#ifndef UART0\_H

#define UART0\_H

#ifndef F\_CPU

#define F\_CPU 8000000UL

#endif

#ifndef BAUDRATE

#define BAUDRATE 115200

#endif

#define ASYNCH\_NORM\_PRESCALER (F\_CPU/16/BAUDRATE - 1)

int USART0\_sendChar(char data, *FILE* \*stream)

/\*

\* Procedure to send a single character over USART0. If character is linefeed, reset

\* line.

\* Assumes ASCII code.

\*/

{

if(data == '\n')

{

while(! (UCSR0A & (1<<UDRE0)) ); // Wait until data is ready to send

UDR0 = '\r'; // send return char.

}

while(! (UCSR0A & (1<<UDRE0)) ); // Wait until data is ready to send.

UDR0 = data; // send data char.

return 0;

}

void USART0\_init (void)

/\*

\* Procedure to initialize USART0 asynchronous with enabled RX/TX, 8 bit data,

\* no parity, and 1 stop bit.

\*/

{

UCSR0B = (1<<TXEN0) | (1<<RXEN0); // enable transmit/receive

UCSR0C = (1<<UCSZ01) | (1<<UCSZ00); // asynchronous, 8N1

UBRR0 = ASYNCH\_NORM\_PRESCALER; // To set 9600 baud rate with 8MHz clock

}

int USART0\_ReceiveChar(*FILE* \*stream)

{

*uint8\_t* u8Data;

// Wait for byte to be received

while(!(UCSR0A&(1<<RXC0))){};

u8Data=UDR0;

// echo input data

//USART0SendByte(u8Data,stream);

return u8Data; // Return received data

}

// reset stream pointer

// Uncomment or place in main. Replace NULL argument with a

// procedure to receive a character if expecting input such as USART0\_ReceiveChar.

//FILE USART0\_stream = FDEV\_SETUP\_STREAM(USART0\_sendChar, NULL, \_FDEV\_SETUP\_WRITE);

#endif // UART0\_H

### i2c.h and i2c.c

Due to the length of these files, to avoid clutter, it has been omitted. It is an I2C library by Peter Fleury (<pfleury@gmx.ch> <http://jump.to/fleury>).

The files can be found here: [i2c.h](https://github.com/martinjaime/CpE301_FinalProject/blob/master/PublishCompass/PublishCompass/i2c.h) and [i2c.c](https://github.com/martinjaime/CpE301_FinalProject/blob/master/PublishCompass/PublishCompass/i2c.c).

### HMC5883L.h

/\*

\* HMC5883L library code

\*

\* I2C Routines

\* 00 Configuration register A R/W 00011100 &H1C

\* 01 Configuration register B R/W 00000000 &H00

\* 02 Mode register R/W 00000000 &H00

\* 03 Data Output X MSB Register Read

\* 04 Data Output X LSB Register Read

\* 05 Data Output Z MSB Register Read

\* 06 Data Output Z LSB Register Read

\* 07 Data Output Y MSB Register Read

\* 08 Data Output Y LSB Register Read

\* 09 Status Register Read

\* 10 Identification Register A Read

\* 11 Identification Register B Read

\* 12 Identification Register C Read

\*/

#define HMC5883L\_ADDR 0x3C // I2C address for device.

#define WRITE\_ADDRESS 0x3C

#define READ\_ADDRESS 0x3D

#define CONF\_A\_REG 0x00 // Addresses for configuration registers.

#define CONF\_B\_REG 0x01 //

#define CONF\_M\_REG 0x02 //

#define X\_MSB\_REG 0x03 // Addresses for MSB of axis registers.

#define Y\_MSB\_REG 0x07 //

#define Z\_MSB\_REG 0x05 //

void HMC5883L\_init()

/\*

\* This function initiates the HMC5883L by setting configuration registers.

\*/

{

// Write 0x70 to HMC5883L register A

// " 0x20 to " " B

// " 0x00 to " " Mode

i2c\_start(HMC5883L\_ADDR+I2C\_WRITE); // start I2C with addcess of HMC5883L

i2c\_write(CONF\_A\_REG); // Write to CONF\_A\_REG

i2c\_write(0x70); // avg 8 samples, 15Hz, Normal measurement

i2c\_write(0x20); // +/-1.3Ga scale.

i2c\_write(0x00); // Continuous measurement mode.

i2c\_stop(); // Release the bus.

*\_delay\_ms*(1); // Delay

}

void readCompass(*int16\_t* \*value)

/\* Given an int16\_t array value of size 3, readCompass will read values stored

\* on each axis register of the HMC5883L and store x, y, and z values at index

\* locations 0, 1, and 2 of value array respectively.

\*/

{

*int16\_t* temp; // temp variable to store current read

i2c\_start(HMC5883L\_ADDR+I2C\_WRITE); // start I2C write with address to HMC5883L

i2c\_write(X\_MSB\_REG); // Begin reading from address X\_MSB\_REG

i2c\_start\_wait(HMC5883L\_ADDR+I2C\_READ); // Read x axis

temp = i2c\_readAck(); // Read X\_MSB

temp = (temp<<8) | i2c\_readAck();//read X\_LSB, OR it with MSB shifted to left by 8

value[0] = temp; // store in value for index X.

// Read z

temp = i2c\_readAck(); // Read Z\_MSB

temp = (temp<<8) | i2c\_readAck();//read X\_LSB, OR it with MSB shifted to left by 8

value[2] = temp; // store in value for index X.

// Read y axis

temp = i2c\_readAck(); // Read Y\_MSB

temp = (temp << 8) | i2c\_readNak();//read X\_LSB, OR it with MSB shifted to left by 8

value[1] = temp; // store in value for index X.

i2c\_stop(); // Release I2C bus.

}

### ESP8266.h

There was an attempt at using AT firmware by using the following custom API.

#ifndef ESP8266\_H

#define ESP8266\_H

// Common responses

#define OK "OK"

#define GOT\_IP "WIFI GOT IP"

// Common commands

#define AT "AT\n"

#define RESET "AT+RST\n"

#include "uart.h"

*int8\_t* ESP8266\_receive(char \*response/\*, char \*badResponse\*/)

// check if response string is received and return 1. Otherwise, return 0.

// https://github.com/mattcallow/esp8266/blob/master/esp8266.ino

{

*uint16\_t* maxTry = 0 - 1; // get max value of uint16\_t;

char ch;

*uint16\_t* i = 0;

// uint16\_t j = 0; // index for bad response.

*uint32\_t* ok\_length = *strlen*(response);

// uint32\_t nok\_length = strlen(badResponse);

while(--maxTry > 0)

{

ch = USART0\_ReceiveChar(*NULL*);

if (ch == response[i])

{

if (++i == ok\_length)

{

return 1; // matched response

}

}

// else if (ch == badResponse[j])

// {

// if (++j == nok\_length)

// return 0; // matched badResponse.

// }

}

return 0;

}

*int8\_t* ESP8266\_command(char \*command, char \*response)

{

*printf*(command);

if (ESP8266\_receive(response))

return 1;

else

return 0;

}

void ESP8266\_init()

{

ESP8266\_command(RESET, OK);

*\_delay\_ms*(100);

ESP8266\_command("AT+CWJAP=\"martinCpE301\_spot\",\"S2016CpE\"\n", GOT\_IP);

}

#endif

## REFERENCE:

[1] Ho-Duck Kim, Sang-Wook Seo, In-hun Jang and Kwee-Bo Sim, "SLAM of mobile robot in the indoor environment with Digital Magnetic Compass and Ultrasonic Sensors," Control, Automation and Systems, 2007. ICCAS '07. International Conference on, Seoul, 2007, pp. 87-90. doi: 10.1109/ICCAS.2007.4406885

[2] HMC5883L datasheet <https://github.com/martinjaime/CpE301_FinalProject/blob/master/Datasheets/HMC5883L.pdf>

[3] HUZZAH ESP8266 datasheet <https://github.com/martinjaime/CpE301_FinalProject/blob/master/Datasheets/adafruit-huzzah-esp8266-breakout.pdf>

[4] AT Instruction set <https://github.com/martinjaime/CpE301_FinalProject/blob/master/Datasheets/AT%20Instruction%20Set__EN_v0.30.pdf>