Wayne's Tinkering Page

Self Driving RC Car >

Steering to a Heading

Human's managed to discover magnetic navigation, in the form of the compass, sometime around the 12th century. However, recently science has discovered that some animals, such as pigeons and sea turtles have know how to do this for much longer. My daughter Belle's 8th grade science fair project investigated whether Madagascar Hissing Cockroaches have the same ability. Interestingly, even in the age of GPS, a self driving car will need a good, reliable compass to help it steer properly.

So, the first navigation task we tackled was how to make the car go in a particular NSEW direction. To run the race course, the car would need to know how to drive through a series of waypoints that define the path through the course. I planned to use a GPS receiver so the car can determine its current location and, using this information, the computer in the car will be constantly computing the distance and bearing (direction) to the next waypoint. But, while GPS receivers are good for determining the car's location in terms of latitude and longitude values, unless the car is moving, GPS receivers are unable to determine which direction the car should drive in order to reach the next waypoint. So, the car needs an additional device, a magnetometer, that it can use determine its orientation in the earth's magnetic field. The human equivalent of this is the compass.

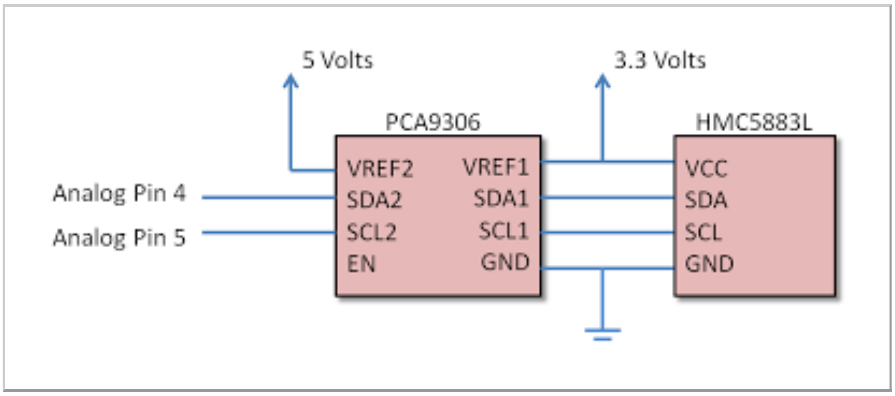
Only a few years back, it was difficult (and/or expensive) to build a digital compass suitable for a self driving RC car, but devices like Apple iPhone that include this function have brought down the price of these devices considerably.

Currently, it's possible to buy a 3 axis digital magnetometer that can be used like a compass for less than $10 in chip form. However, because these devices are used in tight spaces, like cell phones, the IC packages are very small and difficult to solder without professional equipment. So, I elected to purchase a "breakout board" from Sparkfun that ahs the chip already mounted in a small PC board is a form that's very convenient to work with. The device I selected is the Honeywell HMC5883L which costs only $15 when mounted on a Sparkfun on a breakout board.

I picked this device because it's entirely digital and needs only a simple I2C interface to connect to a microcontroller. In addition, there's already an Arduino code library for taking to the device and converting its magnetic readings into a compass heading. However, before trying to get fancy with the device, I decided to wire up a simple test circuit using an Arduino Diecimila and protoboard shield. I usually do this for any new device I work with because I find it helps to get all my stupid mistakes out of the way using a simple circuit before I try to wire up anything more complex. Also, I like to keep the test circuit around during later development because it gives me a quick way to test the device in situations where I might have damaged it due to a wiring mistake in a more complex circuit.

One challenge in using the HMC5883L comes from it being designed to run at 3.3 volts whereas the standard Ardunio runs at 5 volts. One way to solve this would be to use something like the Sparkfun Arduino Mini Pro, which comes in a 3.3 volt variant. However, this version also only runs at 8 MHz rather than the 16 Mhz clock speed of the standard Arduino and I figured I'd need the full clock speed for this task. So, I decided to use a level shifting circuit to interconnect the 3.3 volt HMC5883L to the 5 volt Arduino. However, since the I2C interface is a bidirectional interface, this requires a special type of level shifting circuit that's able to transmit data in both directions on each signal line. Fortunately, Sparkfun makes a nice little circuit board for just this purpose, the PCA9306 Level Translator Breakout board, which costs just $7.

On the Arduino, the I2C interface is implemented by the Wire library, which is hard coded to use two specific I/O pins. On most Arduino boards, SDA (data line) is on analog input pin 4, and SCL (clock line) is on analog input pin 5. On the Arduino Mega, SDA is digital pin 20 and SCL is 21. So, since my test circuit is using a standard Arduino, the connections needed are, as follows:



The code i wrote to test I wrote is derived from an excellent tutorial on the HMC5883L, and companion library. I strongly suggest that you read this tutorial in full before you try to work with the HMC5883L, as it will answer many questions you may have about how a digital, 3 axis magnetometer works and how to use it properly to implement an electronic compass. However, in simplified form, here's the essential code you need to exercise the test circuit present above:

```
#include <Wire.h>
#include <HMC5883L.h>

HMC5883L compass;
float declinationAngle = 0.212;    // Radians

void setup() {
  Serial.begin(57600);
  Wire.begin();
  compass = HMC5883L();
  // Set scale to +/- 1.3 Ga
  int error = compass.SetScale(1.3);
  if (error != 0) // If there is an error, print it out.
    Serial.println(compass.GetErrorText(error));
  // Set measurement mode to continous
  error =
compass.SetMeasurementMode(Measurement_Continuous);
  if (error != 0) // If there is an error, print it out.
    Serial.println(compass.GetErrorText(error));
}

void loop() {
  Serial.print("Heading:\t");
  Serial.print(getDegrees());
  Serial.println(" Degrees   \t");
  delay(100);
}

int getDegrees () {
  // Retrived the scaled values from the compass (scaled to
the configured scale).
  MagnetometerScaled scaled = compass.ReadScaledAxis();
   // Calculate heading when the magnetometer is level, then
```

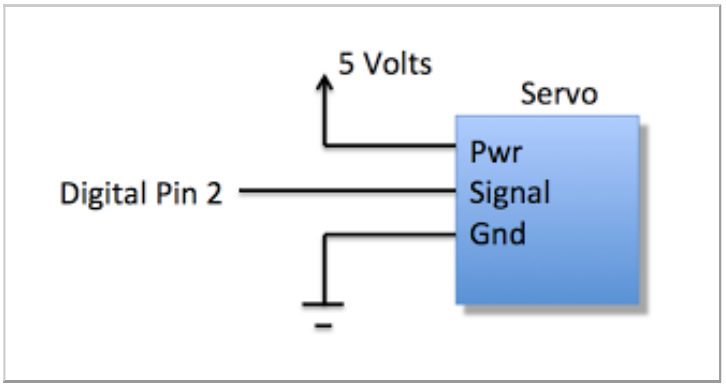
```
correct for signs of axis.
  float heading = atan2(scaled.YAxis, scaled.XAxis) +
declinationAngle;
  // Correct for when signs are reversed.
  if(heading < 0)
    heading += 2 * PI;
  // Check for wrap due to addition of declination.
  if(heading > 2 * PI)
    heading -= 2 * PI;
  // Convert radians to degrees for readability.
  return (int)  (heading * 180 / M_PI);
}
```

If you compile and run this code and then engage the monitor window (set to 57600 baud), you'll see a steady stream of "Heading: nn Degrees" readings where "nn" indicates the direction in decimal degrees (0 - 359) the HMC5883L breakout board is facing relative to the X axis shown on the board. That is, when the X axis is facing magnetic North, the message printed should be "Heading: 0 Degrees". Then, as you rotate the board to the right, this number should increase until you've reached 359 degree, which then wraps around back to 0 degrees. Note: this assumes that the board is held with the Z axis pointing straight up/down and the breakout board is positioned with the HMC5883L chip facing up to the sky.

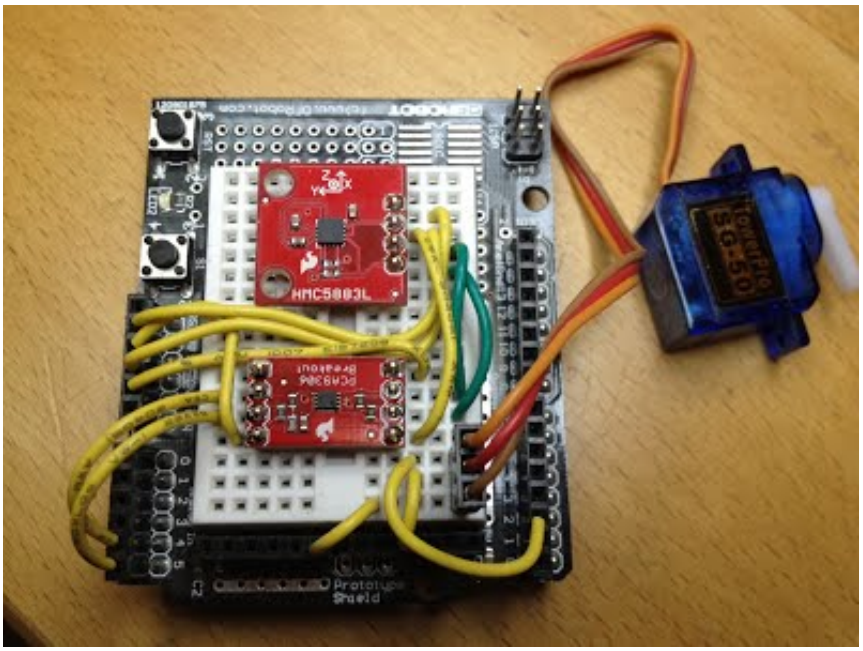
You'll also notice that the code above defines a value called `declinationAngle` that's used to adjust the heading. You should read the article I mentioned above for more information on this but, in essence, it adjust for the fact that magnetic North does not point to true north in many locations, so an adjustment is needed based on your local latitude and longitude. You might also want to read this Wikipedia article, too.

Go North Young Car

The next step is to add in a way for the compass to control the steering servo on the car. The Arduino also has a library called Servo that's intended for just this purpose. In addition to +5 volts and ground, this requires only the use of a single I/O pin to control a typical RC servo through its full range of motion. For my initial tests, I connected a spare servo I had lying around to the compass test circuit I described above using this simple circuit:



The following photo shows the proto board described above with the test servo connected:

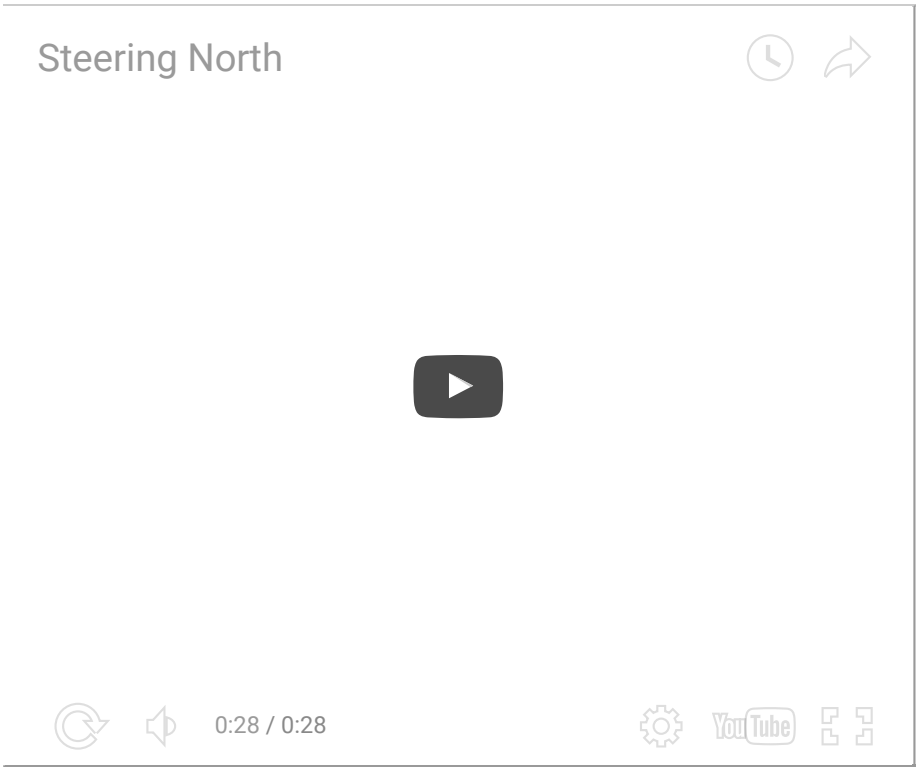


Now, all it takes to implement control of the servo and steer the car in a particular compass direction is to modify the loop() code in a way that compare the car's current heading to a goal setting and then compute whether to move the servo based on the difference. By making a few simple additions to the code above, we get:

```
#include <Wire.h>
#include <HMC5883L.h>
#include <Servo.h>

#define SERVO  2
#define CENTER 100
#define GOAL   0

HMC5883L compass;
int      goal = GOAL;
float    declinationAngle = 0.212;    // Radians
Servo    myservo;


void setup() {
  Serial.begin(57600);
  myservo.attach(SERVO);
  Wire.begin();
  compass = HMC5883L();
  // Set scale to +/- 1.3 Ga
  int error = compass.SetScale(1.3);
  if (error != 0)
    Serial.println(compass.GetErrorText(error));
  // Set measurement mode to continous
  error =
compass.SetMeasurementMode(Measurement_Continuous);
  if (error != 0)
    Serial.println(compass.GetErrorText(error));
}

void loop() {
  int angle = getDegrees();
  int error = goal - angle;
  if (error >= 180)
```

```
    error -= 360;
  if (error <= -180)
    error += 360;
  // Update servo and keep with range of +/- 60
  if (error > 60)
    error = 60;
  if (error < -60)
    error = -60;
  myservo.write(CENTER + error);
  delay(10);
}

int getDegrees () {
  MagnetometerScaled scaled = compass.ReadScaledAxis();
   // Calculate heading when the magnetometer is level, then
correct for signs of axis.
  float heading = atan2(scaled.YAxis, scaled.XAxis) +
declinationAngle;
  // Correct for when signs are reversed.
  if(heading < 0)
    heading += 2 * PI;
  // Check for wrap due to addition of declination.
  if(heading > 2 * PI)
    heading -= 2 * PI;
  // Convert radians to degrees for readability.
  return (int)  (heading * 180 / M_PI);
}
```

The value for GOAL is defined as 0, so this means the car will, as the section title said, always try to steer the car North. In practice, the above code will probably only steer the car close to North because the error signal that controls the movement of the servo gets smaller as the car steers closer to North and these smaller values are not enough to move the car that last little bit to its goal. The solution for this is a more sophisticated type of control loop called a PID controller (more about this later.) However, as a starting point, the above code does a pretty good job, as demonstrated in the following video.



The video above was made with the actual Arduino program listed above controlling the steering mechanism on the car while I controlled the throttle manually. After I drove the car downrange (South), I switched the steering to program control and, when I pushed the throttle, the car turned around and drove North. A Failsafe Mux board made by DIY Drones was used to toggle control of the steering back and forth between the Arduino and the RC transmitter. This is actually a great accessory to have when working on a computer controlled car, as you never know when the next robot uprising will occur...

Next: Navigating with GPS