.NAV

Navigational Autonomous Vehicle

Joseph Varikooty

Ryan Wallace

University of Illinois

I. Introduction

- Problem description

  Our team has constructed a vehicle that will traverse on a path indicated by longitudinal and latitudinal coordinates. The chosen location where the vehicle would navigate around was the parking lot directly to the east of Beckman Institute *(map of route in Appendix)*; chosen for its close proximity to the Electrical and Computer Engineering Building. Although, any location which offers a clear GPS signal would have sufficed. Our vehicle checked current location, current heading and compared these values to the desired location and desired heading 10 times per second to either turn the vehicle in the direction of the proper heading and light the corresponding yellow turn LED or continue on its current path and light the green LED until the next GPS location is reached. After the vehicle traveled to all of the given GPS coordinates it would then stop at the final coordinate. The red LEDs would begin flashing and display the group member's names on the LCD.

- Design concept

  In our GPS navigation car design, a GPS module and a compass module determined input and output signals through the arduino and h-bridge motor driver in order to control the speed of the corresponding motors while navigating from waypoint to waypoint. The GPS module was used to obtain current GPS coordinates of the robot. The compass module detects current heading. With current GPS location and current heading, the GPS location is then compared to the coordinates of the desired location of the next waypoint. Once this comparison is made the compass module's current position is checked with the proper heading needed to travel to the desired location, and would then alter corresponding motor speed until the heading aligned with the direction of the next waypoint.

The following table describes the various current states of the vehicle and corresponding action taken:

Current State	**Action**
Vehicle at waypoint: W_n (n= 0,1,2,3,...)	Vehicle should navigate to waypoint: W_{n+1}
Vehicle at final waypoint: W_f	Vehicle should stop and display final message on LCD. Red LEDs blink
Vehicle's heading is pointed to the left of the next waypoint	Car turns right (motor controlling left wheel is increased while motor controlling right wheel is decreased). Yellow (right side) LED illuminates
Vehicle's heading is pointed to the right of the next waypoint	Car turns left (motor controlling right wheel is increased while motor controlling left wheel is decreased). Yellow (left side) LED illuminates
Vehicle's heading is pointed in the direction of the next waypoint	No speed variation in the motors both motors drive the vehicle to the next location. Green LED illuminates

Table 1: Current States and Corresponding Behaviors for Navigating Vehicle

The following is a table of items used to build our navigating vehicle:

Item	Value (if applicable)	Quantity
Adafruit GPS Module (MTK3339 - based)		1
Arduino Motor Shield (L298P)		1
Arduino Uno		1
Compass Module (HMC5883L)		1
DC Motors (from Sparkfun Chassis Kit)		2
Green LED		1
Lithium Ion AA Batteries	1.5 V	4
Lithium Ion AAA Batteries	1.5 V	4
Parallax Serial LCD Screen		1
Red LED		2
Vehicle Chassis Kit (from Sparkfun)		1
Yellow LED		2

Table 2: Items Needed for the Navigating Vehicle

II. Analysis of components

Adafruit GPS Module (MTK3339-based):
Device that can track GPS satellites and compute its latitude and longitude location.

Characterization of GPS Module:
The GPS module outputs a standard NMEA sentence over serial communication. These are comma separated strings that all gps modules output. We tell the gps module to output both the GGA sequence and RMC sequence.

GGA - essential fix data which provide 3D location and accuracy data	
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47	
GGA	Global Positioning System Fix Data
123519	Fix taken at 12:35:19 UTC
4807.038,N	Latitude 48 deg 07.038' N
01131.000,E	Longitude 11 deg 31.000' E
1	Fix quality: 0 = invalid 1 = GPS fix (SPS) 2 = DGPS fix 3 = PPS fix 4 = Real Time Kinematic 5 = Float RTK 6 = estimated (dead reckoning) (2.3 feature) 7 = Manual input mode 8 = Simulation mode`
08	Number of satellites being tracked
0.9	Horizontal dilution of position
545.4,M	Altitude, Meters, above mean sea level
46.9,M	Height of geoid (mean sea level) above WGS84
(empty field)	time in seconds since last DGPS update
(empty field)	DGPS station ID number

*47	the checksum data, always begins with *

RMC - The Recommended Minimum	
$GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A	
RMC	Recommended Minimum sentence C
123519	Fix taken at 12:35:19 UTC
A	Status A=active or V=Void.
4807.038,N	Latitude 48 deg 07.038' N
01131.000,E	Longitude 11 deg 31.000' E
022.4	Speed over the ground in knots
084.4	Track angle in degrees True
230394	Date - 23rd of March 1994
003.1,W	Magnetic Variation
*6A	The checksum data, always begins with *

Design Considerations:
The GPS module was mounted on an elevated breadboard that was lined with copper tape to protect it from electromagnetic interference from the motors. It was also important to mount the module such that it has a line of sight with the sky and gps satellites.

*Referenced from: http://www.gpsinformation.org/
**GPS Datasheet Link in Appendix

HMC5883L Compass Module:
Honeywell's 3-axis digital magnetometer. Senses the magnetic field strength in the x, y, and z axis axis and communicates the data over I2C.

Characterization of Compass Module:

The HMC5883L communicates via a two wire I2C bus. The data acquired by this device is stored in a number of on-chip 8-bit registers. These registers are read, appropriately shifted, and the combined raw data is stored by the microcontroller. This raw data then needs to be multiplied by a user set gain value. After this, readings in milliGauss for each axis is obtained.

Design Considerations:
This device is extremely sensitive to external magnetic fields. Thus, it was mounted on an elevated breadboard that was lined with copper to tape to shield it from electromagnetic interference from the motors and any other sources of significant current draw.

**HMC5883L Datasheet Link in Appendix

Parallax Inc. Serial LCD Module:
A green backlit four row, twenty characters per row LCD module that we commute to via serial command.

Characterization of LCD Module:
Messages are written on this display by first writing a cursor position and then the desired ASCII characters. A serial baud rate of 19200 bps was chosen to ensure fastest execute time of the serial write code.

Design Considerations for LCD Module:
The LCD screen was mounted on the back of the vehicle and angled upward. This was to ensure good viewing angle and distance between the screen and the noise sensitive gps and compass modules.

**Parallax LCD Datasheet Link in Appendix

Arduino Motor Shield (L298-based):
A dual full-bridge driver designed to drive inductive loads. This shield is based the L298 chip which can drive up to 46V and 4A (2A per channel). A current sense ability is also included.

Characterization of Motor Shield:
This shield has two separate channels, called A and B, that each use four of the I/O pins to drive or sense the motor.
The pins for the shield are shown in the table below:

Function	**Pins for Ch. A**	**Pins for Ch. B**
Direction	D12	D13
PWM	D3	D11
Brake	D9	D8
Current Sensing	A0	A1

Note: The current sense feature was not used in this project so it was not characterized.

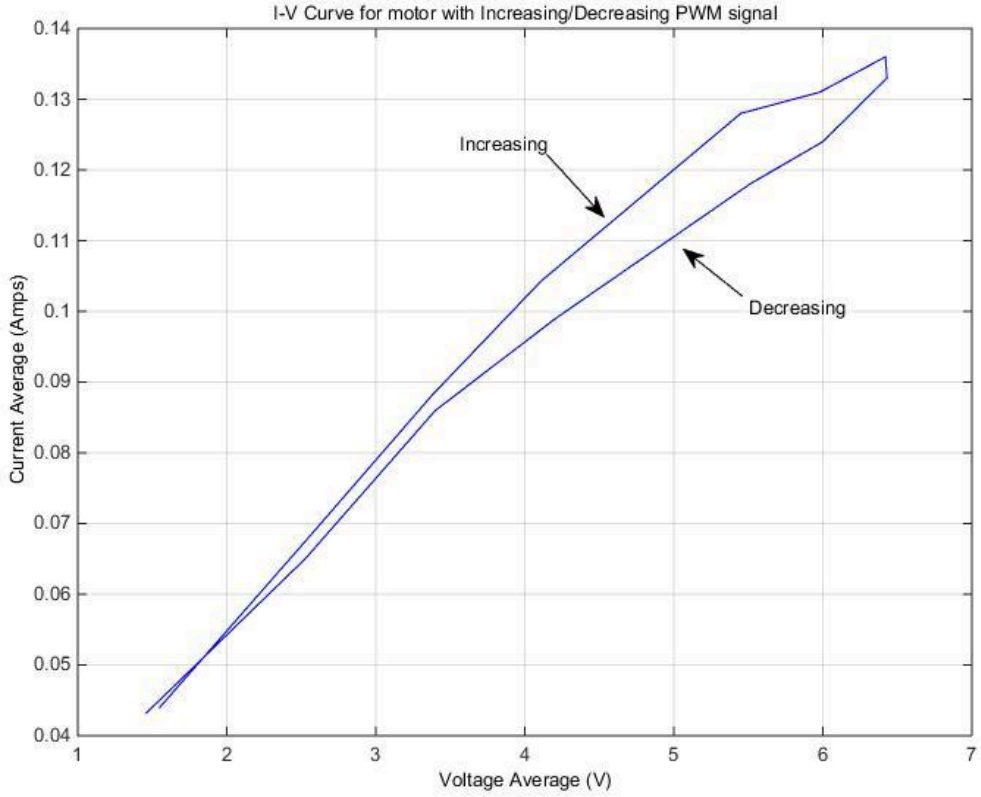
Design Considerations for Motor Shield:
The motor shield piggybacks on top of the arduino board. It was important to give this board good airflow because the L298 chip can get fairly hot under load.

**The L298 Datasheet Link in the Appendix

Motors:
This is the component that was used to turn the wheels of our robotic vehicle. With power applied the motor would turn a shaft that was attached to the wheels.

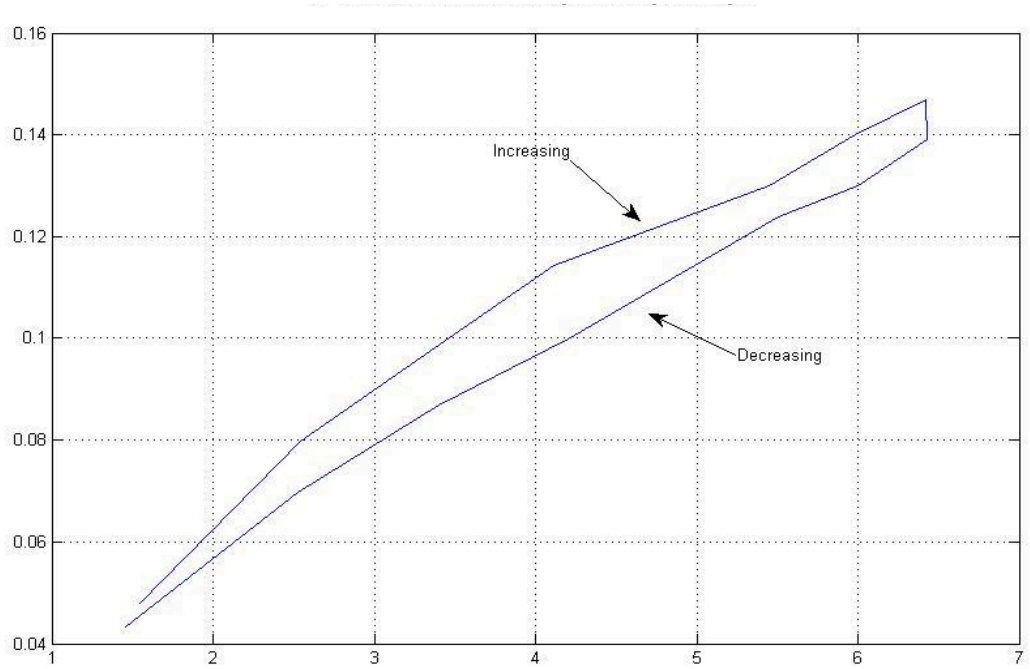
Characterization of Motor:

**Note the offset of the curves in the graphs on the next page. The right motor draws more current and rotates faster compared with the left motor when the same PWM signal fed.



Left motor characterization (above)*F1 Right motor characterization (below)*F2

*F1 & *F2: Figure 1 & 2 for the IV curves of PWM increasing/decreasing power in our motors



Voltage Average vs Current Average of the motor for an increasing and decreasing PWM signal is characterized in the graph on the previous page. Using a function generator to produce our PWM signal we connected one DMM (digital multimeter) on either side of our motor to read the voltage (V_{avg})and a second DMM in series with the motor to measure the current (I_{avg}). Data points for this graph were obtained by starting our duty cycle at 20%, increasing to 80%, decreasing back down to 20%, and taking measurements at each 10% increment. This graph indicates a relatively linear I-V curve compared to that of the non-PWM design which had a stall region which accounted for a spike in current draw and a less gradual motor run time.

Design Considerations:

The PWM design was chosen as opposed to the non PWM design which utilized a constant voltage applied to a varying resistance in series with the motor, the reason being the difference between the I-V characteristics of the two circuits in regards to the stall region of both designs. The non PWM design had a larger current draw leading up to motor turn on due to more energy required in overcoming the internal resistance of the stalled motor. As opposed to the PWM design that did not result in this spike in current draw/stall time. Had we gone with the non PWM design there was potential for unwanted stall times. Also note that each of our two motors was slightly different and thus gave different RPM speeds when the same PWM signal supplied. We had to compensate for this problem by adding a unique scaling factor for each motor in the code.

III. Design Description

Block Diagram:

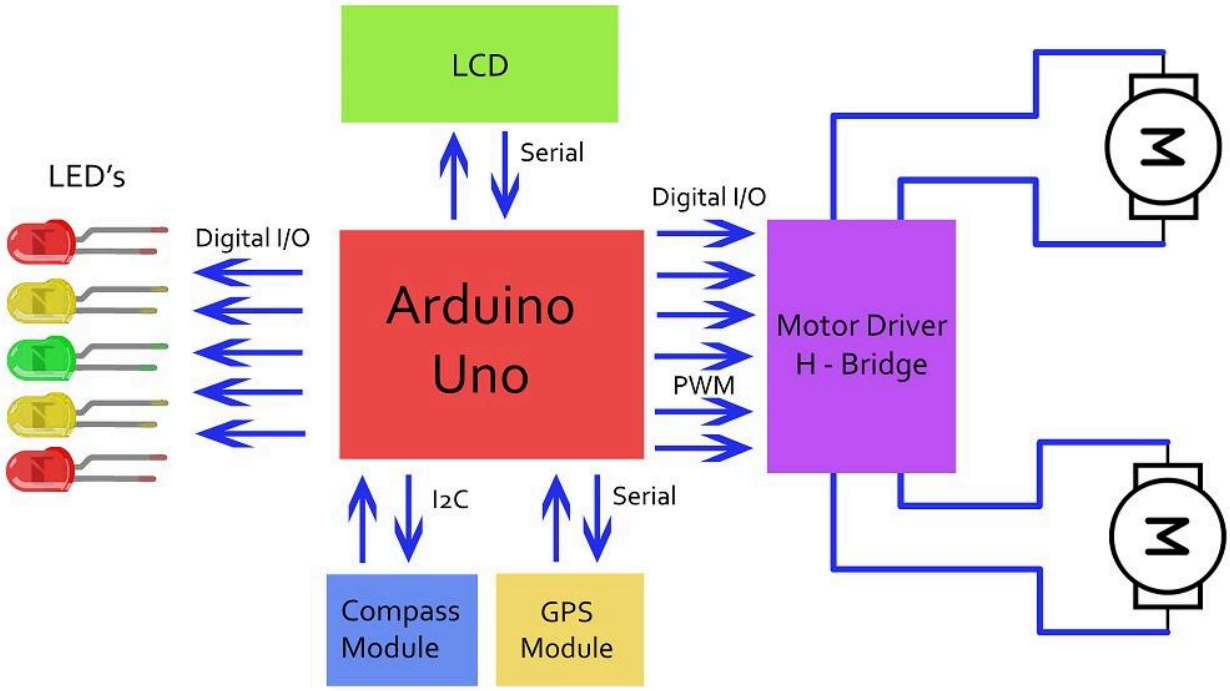


Figure 3: Block Diagram of Design

Arduino Code:

**Full code in Appendix

Pseudo Code / general code overview:

```
void setup(){                    //this code runs only once when the arduino first starts

initiate_compass_sensor();       //set our gain and tell it to continuously output data

if(!compass_detected){
        write_to_LCD;            //write that our compass is not connected
        loop_forever();          //stall program. We cannot continue without the compass
}

tell_GPS_to_output_GGA_and_RMC_NMEA_sentences();

while(!GPS_is_aquired){          //when we haven't acquired gps yet...
        write_to_LCD;            //write that we are waiting for GPS signal
```

```
}

void loop(){            //this code is looped continuously after setup() is run

record_our_start_time();   //used later to make our code run at 10Hz

if (we_received_a_new_NMEA_string){
        parse_NMEA_string();       //extract and store the data we want
}

get_current_waypoint_lat_long();         //load current waypoint's lat and long

calculate_distance_between_where_we_are_and_waypoint();

if(distance_between_where_we_are_and_waypoint <= distance_tolerance){
        waypoint#++;         //increment to next stored waypoint
        if(we_reached_our_last_waypoint){
                break_motors();              //stop the motors from spinning
                write_to_LCD();              //write that we've reached last waypoint
                while(1){                    //loop forever
                        flash_red_leds();
                }
                }
        get_current_waypoint_long_lat();         //load current waypoint's lat and long
}

calculate_current_compass_heading();

calculate_desired _heading_from_our_current_lat_long_to_current_waypoint();

calulate_error_between_current_heading_and_desired_heading();

if(we_need_to_go_straight){
        turn_on_green_led();
}
if(we_need_to_go_left){
        turn_on_left_yellow_led();
}
if(we_need_to_go_right){
        turn_on_right_yellow_led();
}

calculate_speed_for_each_motor_to_turn_to_heading();
 /*We use our calculated heading error and feed it to a PD (proportional derivative)
control code. To go left portionately slow down the left motor and increase speed of
```

```
right motor and vice versa to turn right. We use derivative control to stall oscillations
from occurring*/

write_to_motor_shield();  //send PWM speed and motor direction

write_to_LCD;          //write current vehicle information

delay(100ms - (current_time - start_time));       //make our code runs at 10Hz
/* Code is run at 10Hz to ensure consistent and accurate motor correction. */
}
```

Circuit Schematic:

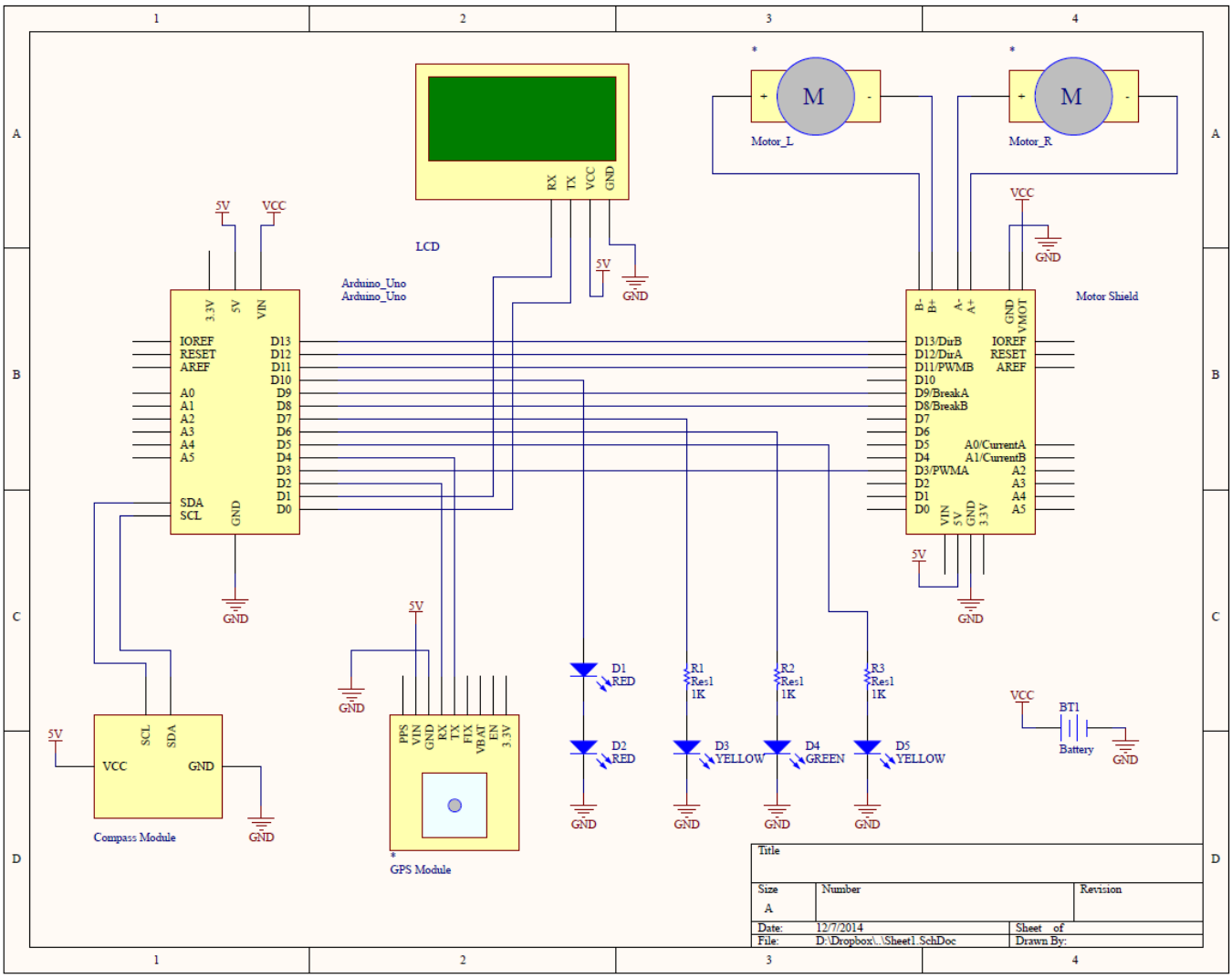


Figure 4: Circuit Diagram for design

Mechanical Construction:

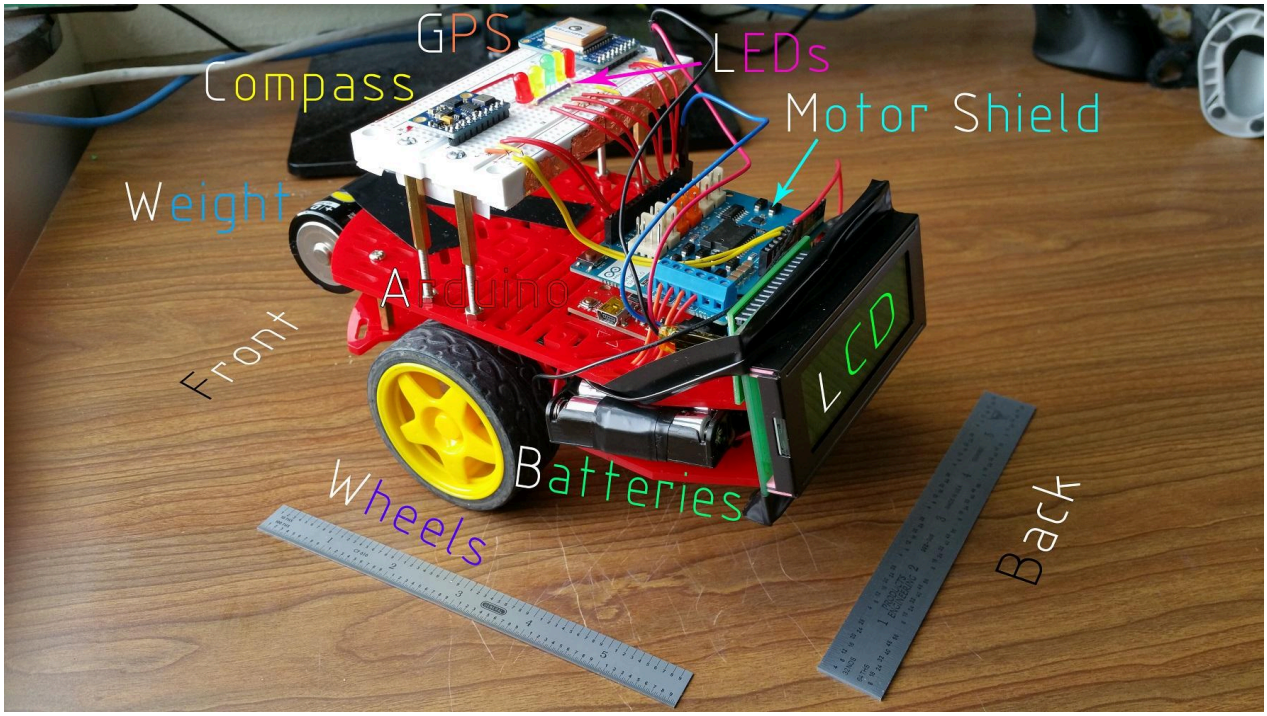
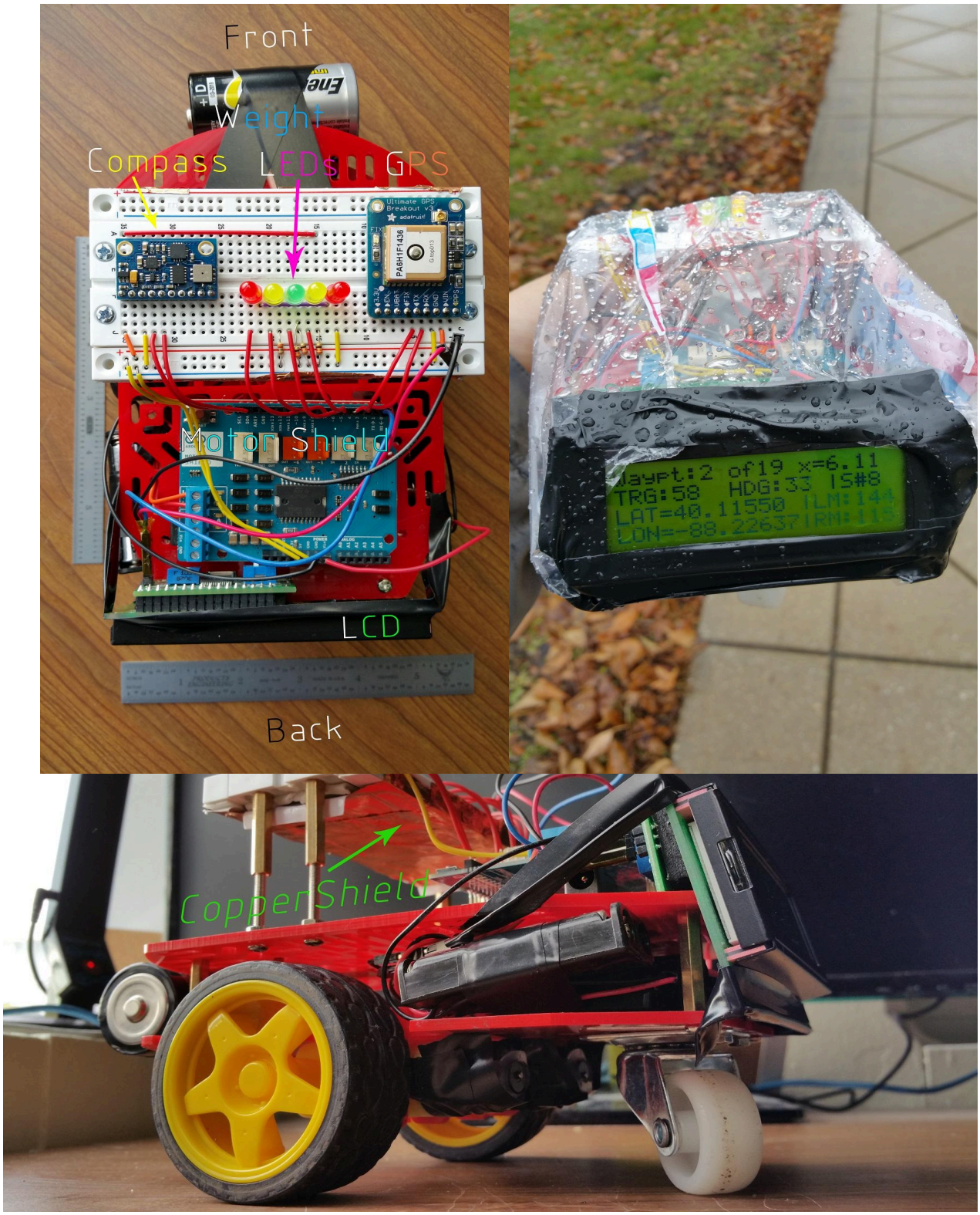


Figure 5: Navigational Vehicle physical appearance

As seen in the figure 5, the GPS module and compass module are mounted to a protoboard which was raised off of the chassis using various hardware from the Sparkfun kit. The LCD directed towards the back of the vehicle and provided a real time readout of current location, distance from next location and proper heading amongst other outputs which allowed us to troubleshoot the vehicle while moving.



Clockwise starting from upper left: Figure 6, Figure 7, Figure 8

As seen in figure 6 on the previous page the LEDs located at the top of the vehicle provided an extra bit of troubleshooting assistance and all around more aesthetic look to the vehicle. During presentation the 3 interior LEDs indicated the direction with which the vehicle needed to turn in order to achieve a proper heading and the 2 outer red LED's flashing at the end of a completed run signaled success.

Figure 7 on the previous page provides a glimpse into some of the environmental challenges we faced, a makeshift raincoat (a clear bag) as seen here was used to keep our circuit dry during testing. The LCD screen as shown here provided clear readout of: current coordinates, the number of waypoints the vehicle has currently gone through, the total number of waypoints, the distance between current coordinate and next waypoint, the target heading, the current heading and finally how many GPS satellites our vehicle was currently connected to.

Figure 8 on the previous page provides a slightly different angle of the vehicle, from this viewpoint you can see the copper shielding placed under the protoboard in order to prevent interference from the electrical components located below. This protoboard was raised 3 inches from the top of the chassis. A clear shot of the front wheel as opposed to the ball bearing wheel that came with the Sparkfun kits can also be seen in this photo.

IV. Conclusion

- Lessons learned

When testing our compass and GPS modules, early tests proved that they picked up interference when directly next to the power source (battery pack) and motors. During early construction we relocated the GPS and compass modules by mounting them to a protoboard and raising the protoboard 3 inches off of the chassis and covering the bottom of this protoboard with copper shielding used to help keep interference from electrical components to a minimum. Outdoor testing also proved difficult with the front wheel provided in the Sparkfun kits, the ball bearing wheel provided picked up too much debris and soon began to cause unwanted friction. The ball bearing wheel was changed for a standard white wheel as pictured previously in figure 8. Initial testing was done on a relatively warm night for November (approximately 50° fahrenheit) with clear skies. The location where the testing was conducted was approximately 5 miles south-west of the Electrical and Computer Engineering Building. An open parking lot surrounded by a residential area with no large structures within close proximity. The vehicle was powered by a 7.2 V NiMh rechargeable battery. The vehicle navigated through the waypoints in the proper order until reaching the last waypoint and stopping (LEDs/LCD were not added at this point). Further testing on the large ellipse shaped sidewalk located directly to the east of the ECE building was conducted with much different results. In slicker conditions (rain and ice) the vehicle's grip seemed to heavily affect motor control. The vehicle began to lose power shortly into test runs and the navigation seemed to be much more erratic. We determined that the environment was now our main obstacle in getting the vehicle to navigate properly. The temperature had dropped significantly since the initial test run, dropping now into the teens and even single digits with windchill. Our NiMh battery's current was beginning to drop significantly under these new temperatures and was not supplying the current needed to properly power the vehicle. The buildings and structures surrounding the area were also much larger, obstructing a proper GPS signal and constantly changing the location of waypoints. After adding a weight to add extra friction for the wheels which alleviated the grip issue we began to troubleshoot the battery and considering possible alternatives. After researching batteries that would perform in the cold we narrowed our search to lithium ion batteries, which were rated to -40° fahrenheit before deviation in current/voltage ratings occurred. As a backup to this plan we made a portable heating device using a 2 foot by 1 foot section of

an emergency heat blanket along with a hand warmer and wrapped the NiMh battery and hand warmer within the emergency blanket. Our final design utilized eight 1.5 volt lithium ion batteries wired in series for a total of 12 volts. This provided ample power and withstood the cold weather conditions. In order to resolve the GPS signal issues we began performing test runs with the vehicle in the parking lot directly to the east of Beckman Institute. Although there were still some occasional hiccups when the vehicle approached the Beckman Institute. The parking lot was more of an open area with less large structures surrounding it and GPS signal seemed more accurate when in this location as well. Further troubleshooting became much more simplified with the addition of an LCD screen and LED lights. With the aid of the LCD screen we were now able to view the number of GPS satellites our vehicle was successfully connected to, the distance and direction of the next waypoint all in real time. The LEDs provided visible indications as to what direction the vehicle wanted travel in order to reach the next waypoint. These components allowed us to narrow down whether our vehicle was navigating improperly due to motor issues or if there were GPS issues. GPS issues included the vehicle determining a waypoint in a much different location than coordinates input.

- Self-assessment

  The navigational vehicle operated as expected, all of the vehicle's operations were in accordance to our current state table. Despite weather conditions and nearby large structures our vehicle was able to navigate through designated waypoints and circle the parking lot avoiding parked cars and interior curbs. The 8 lithium ion batteries performed successfully at powering the vehicle in the cold weather conditions during the presentation. The LED's illuminated properly to indicate direction, along with the blinking red LED's to indicate the last waypoint was reached. These LED's in conjunction with the LCD screen proved our greatest troubleshooting devices as predicted and all LCD commands were properly output during our presentation as well, including the display of both team members name's at the arrival of the final waypoint. Overall the project was a great success.

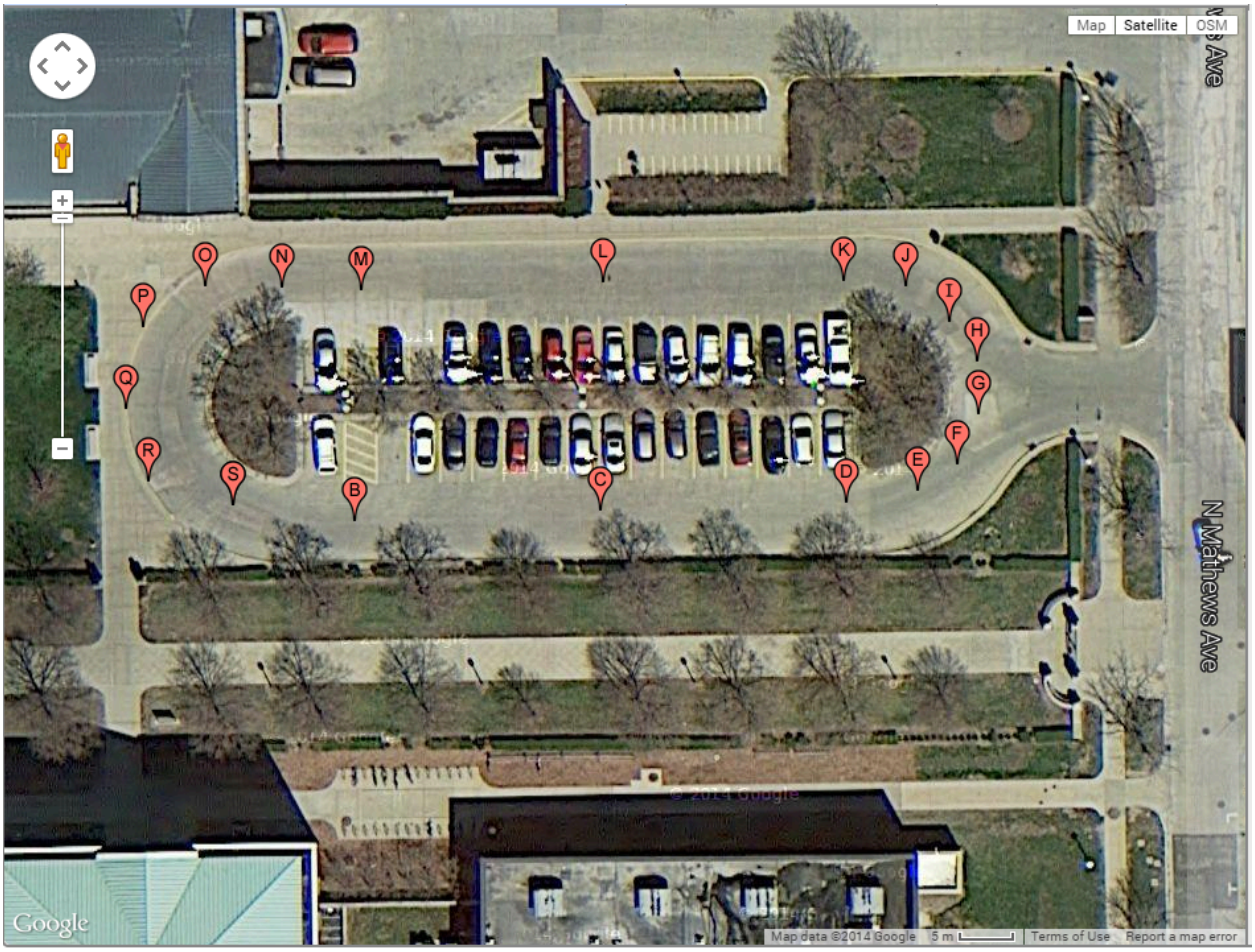
- Future work

  We are interested in adding an obstacle detection aspect in conjunction with a reroute feature. By placing 3 forward facing sonar sensors at the front of the vehicle, one facing straight ahead and two others facing opposing 45° angles

from the straight ahead reference angle. When an object is encountered all 3 sensors send data which will adjust corresponding motors in order to avoid the obstacle. For instance if an object say a curb with a height of roughly 7 inches is approached data is first gathered which characterizes the layout of the curb, whether there is no curb detected towards the right or left will determine motor control in order to turn the vehicle. If for instance an object is sensed by all 3 motors the motor controlling the left wheel is increased while the motor controlling the right wheel is decreased in order to avoid the object. This motor variation will be carried out until there is no longer data of an impending object by the forward and rightward facing sonar sensors, at this point the vehicle will travel forward until there is no longer data of an impending object read by the left facing sensor. Once there is no longer an object sensed by the left facing sensor for a time period of 2 seconds, the vehicle's motor controlling right wheel is increased while motor controlling left wheel is decreased to allow for a proper turn. The same system of events is to be carried out when there is initially no data of an impending object read only by the left facing sensor with opposite motor steering effects as those previously mentioned. This would allow the vehicle to avoid a majority of commonly seen road obstacles but would have potential issue for obscure road hazards such as large potholes or waypoints that are located in an area that require crossing over a curb. Taking these conditions into consideration would exponentially raise the level of difficulty of the code written to the arduino in addition to new components to sense and maneuver such obstacles. The use a control unit which stores a large variety of current/next states and is able to take input/output of data in order to convey proper next state is also a viable solution for a much larger project.

V. Appendix

Map of route traveled by the vehicle:



Link to GPS Datasheet:
http://www.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf

Link to HMC5883L Datasheet:
http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf

Link to Parallax LCD Datasheet:
http://www.parallax.com/sites/default/files/downloads/27979-Parallax-Serial-LCDs-Product-Guide-v3.1.pdf

Link to L298 Datasheet:
https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf

Full Code:

Link:
https://www.dropbox.com/s/xr172059t3egd2p/AutoNav_3_1_working1.ino?dl=0

Also attached below here:

```
#include <Wire\Wire.h>
// I2C Communication
#include <Adafruit_Sensor\Adafruit_Sensor.h>                        // Sensor Abstraction
Layer
#include <Adafruit_HMC5883_Unified\Adafruit_HMC5883_U.h>   // Compass
#include <Adafruit_GPS_Library\waypointClass.h>                          // Custom
class to manaage GPS waypoints
#include <Adafruit_GPS_Library\Adafruit_GPS.h>                           // GPS
#include <SoftwareSerial\SoftwareSerial.h>                               // GPS
#include <math.h>
 // Distance / Heading Calculation

//#define DEBUG                                 //USB debug mode; uses Serial Port,
displays diagnostic information, etc.
#define LCD                                           //use serial LCD to display
diagnostic information
#define LED                                           //use LED's to indicate turn
direction
//#define Move                                  //uncomment to allow the motors to move

// Setup magnemeter(compass)
Adafruit_HMC5883_Unified compass = Adafruit_HMC5883_Unified(12345);
sensors_event_t compass_event;

// Compass navigation
int targetHeading;              // where we want to go to reach current waypoint
int currentHeading;             // where we are actually facing now
int headingError;               // signed (+/-) difference between targetHeading and
currentHeading
#define HEADING_TOLERANCE 10     // tolerance +/- (in degrees) within which inicate that
the robot is going straight

// GPS Navigation
#define GPSECHO false           // set to TRUE for GPS debugging if needed
//#define GPSECHO true          // set to TRUE for GPS debugging if needed
#define rxPin 4                               //SoftwareSerial rxPin
#define txPin 2                               //SoftwareSerial txPin
SoftwareSerial mySerial(rxPin, txPin);
Adafruit_GPS GPS(&mySerial);
boolean usingInterrupt = false;
float currentLat,
```

```
currentLong,
targetLat,
targetLong;
float distanceToTarget,                      // current distance to target (current
waypoint)
originalDistanceToTarget;            // distance to original waypoing when we started
navigating to it


// Waypoints
#define WAYPOINT_DIST_TOLERANCE  3  // tolerance in meters to waypoint; once within this
tolerance, will advance to the next waypoint
#define NUMBER_WAYPOINTS 18         // enter the numebr of way points here (will run
from 0 to (n-1))
int waypointNumber = -1;                           // current waypoint number; will run
from 0 to (NUMBER_WAYPOINTS -1); start at -1 and gets initialized during setup()
waypointClass waypointList[NUMBER_WAYPOINTS] = { /*waypointClass(40.11554, -88.22667),*/
waypointClass(40.11553, -88.22655), waypointClass(40.11554, -88.2263),
waypointClass(40.11554, -88.22605), waypointClass(40.11555, -88.22597),
waypointClass(40.11557, -88.22593), waypointClass(40.11561, -88.22591),
waypointClass(40.11565, -88.22591), waypointClass(40.11568, -88.22594),
waypointClass(40.11571, -88.22599), waypointClass(40.11572, -88.22605),
waypointClass(40.11571, -88.22629), waypointClass(40.11571, -88.22654),
waypointClass(40.11571, -88.22662), waypointClass(40.11571, -88.2267),
waypointClass(40.11568, -88.22676), waypointClass(40.11562, -88.22678),
waypointClass(40.11556, -88.22676), waypointClass(40.11554, -88.22667) };

// LED indictors for steering/turning
/*
enum direction { left, right, straight };
direction turnDirection = straight;
*/
#define straight 6     //pin# for led indictating we are going straight
#define left 7         //pin# for led indictating we are going left
#define right 5        //pin# for led indictating we are going right
#define end 10         //pin# for led indictating we have reached our last waypoint

//Motor Control (L298 Motor Shield)
#define PWM_Right 3
#define PWM_Left 11
#define DIR_Right 12
#define DIR_Left 13
#define BRAKE_Right 9
#define BRAKE_Left 8
#define SNS_Right A0
#define SNS_Left A1
#define motorRIGHTmaxFOWARD 218 //214 //240
#define motorLEFTmaxFOWARD 227 //255
#define maxMotorSpeed 210 //227 //255
int motorspeedL;
int motorspeedR;

//PD control
int last_error;
unsigned long start_time;
int PD_output;
int motorCorrection;
```

```
#define baselineMotorSpeed 150                              //Set the baseline Motor
Speed
int maxCorrection = maxMotorSpeed - baselineMotorSpeed;

// Interrupt is called once a millisecond, looks for any new GPS data, and stores it
SIGNAL(TIMER0_COMPA_vect)
{
 GPS.read();
}

// turn interrupt on and off
void useInterrupt(boolean v)
{
 if (v) {
        // Timer0 is already used for millis() - we'll just interrupt somewhere
        // in the middle and call the "Compare A" function above
        OCR0A = 0xAF;
        TIMSK0 |= _BV(OCIE0A);
        usingInterrupt = true;
 }
 else {
        // do not call the interrupt function COMPA anymore
        TIMSK0 &= ~_BV(OCIE0A);
        usingInterrupt = false;
 }
}

void setup(){
 // Configure the Motor Shield outputs
 pinMode(BRAKE_Right, OUTPUT); // Brake pin on channel A
 pinMode(DIR_Right, OUTPUT);   // Direction pin on channel A
 pinMode(BRAKE_Left, OUTPUT);  // Brake pin on channel B
 pinMode(DIR_Left, OUTPUT);    // Direction pin on channel B
 // Set LED pins as outputs
 pinMode(straight, OUTPUT);
 pinMode(left,OUTPUT);
 pinMode(right,OUTPUT);
 pinMode(end,OUTPUT);

#ifdef DEBUG
 // turn on serial monitor
 Serial.begin(115200);               //required speed for GPSs
#endif

#ifdef LCD
 Serial.begin(19200);            // use a baud rate of 19200 bps
 //pinMode(1, OUTPUT);           // set pin1 as an output (pin1=TX)
 Serial.write(12);               // clear screen & move to top left position
 Serial.write(17);                           // turn on backlight
 delay(5);                                          // required delay
#endif

 // start Mag / Compass
 if (!compass.begin()){
#if defined(DEBUG) || defined(LCD)
        Serial.println(F("COMPASS ERROR!"));
#endif
```

```
        loopForever();         // loop forever, can't operate without compass
 }
 // start GPS and set desired configuration
 GPS.begin(9600);                                    // 9600 NMEA default speed
 GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);   // turns on RMC and GGA (fix data)
 GPS.sendCommand(PMTK_SET_NMEA_UPDATE_10HZ);      // 10 Hz update rate
 GPS.sendCommand(PGCMD_NOANTENNA);                // turn off antenna status info
 useInterrupt(true);                                 // use interrupt to constantly pull
data from GPS
 delay(1000);

 // Wait for GPS to get signal
#if defined(DEBUG) || defined(LCD)
 Serial.println(F("Waiting for GPS"));
#endif
 unsigned long startTime = millis();
 while (!GPS.fix)                        // wait for fix, updating display with each new
NMEA sentence received
 {
#if defined(DEBUG) || defined(LCD)
#ifdef LCD
        Serial.write(168);    //move to line 2, pos 0
#endif
        Serial.print(F("Wait Time: "));
#ifdef LCD
        Serial.write(179);    //move to line 2, pos 11
#endif
        Serial.println((int)(millis() - startTime) / 1000);     // show how long we
have waited
        delay(500);
#endif
        if (GPS.newNMEAreceived())
               GPS.parse(GPS.lastNMEA());
 }
#if defined(DEBUG) || defined(LCD)
#ifdef LCD
 Serial.write(12);
 delay(5);
#endif
 Serial.println(F("GPS Acquired!"));
 Serial.println(F("Starting in..."));
#endif
 for (int i = 10; i > 0; i--)
 {
#if defined(DEBUG) || defined(LCD)
#ifdef LCD
        Serial.write(188);
#endif
        Serial.println(i);
        Serial.println(F(" "));
#endif
        if (GPS.newNMEAreceived())
               GPS.parse(GPS.lastNMEA());
        delay(100);
 }
 nextWaypoint();
}
```

```
void loop(){
 start_time = millis(); //record start time
 // process GPS
 if (GPS.newNMEAreceived())               // check for updated GPS information
 {
        if (GPS.parse(GPS.lastNMEA()))      // if we successfully parse it, update our
data fields
               processGPS();
 }

 // navigate
 currentHeading = readCompass();    // get our current heading
 calcDesiredTurn();                // calculate turn

 // move robot
 move(PDcontrol(headingError));

 //print debug statements
#ifdef DEBUG
 printdebug();
#endif

 //LCD
#ifdef LCD
 printLCD();
#endif
 delay(100 - (millis() - start_time));  //figure out how long loop took to run, subtract
from 100, and delay amount to make loop run 10hz
}

// Loops forever
void loopForever(void){
 while (1)
        ;
}

// Display free memory (SRAM)
#ifdef DEBUG
int freeRam()
{
 extern int __heap_start, *__brkval;
 int v;
 return (int)&v - (__brkval == 0 ? (int)&__heap_start : (int)__brkval);
}
#endif

// reads the magnetometer and outputs a true heading in degrees
int readCompass(void)
{
 compass.getEvent(&compass_event);
 float heading = atan2(compass_event.magnetic.y, compass_event.magnetic.x);

 // Once we have our heading, we must add 'Declination Angle', which is the 'Error' of
the magnetic field in our location.
 // Found here: http://www.ngdc.noaa.gov/geomag-web/#declination
 // Magnetic declination for UIUC: 3° 2' 19" WEST;  1 degreee = 0.0174532925 radians
```

```
#define DEC_ANGLE 0.053756141
 heading += DEC_ANGLE;

 // 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.
 float headingDegrees = heading * 180 / M_PI;

 return ((int)headingDegrees);
}  // readCompass()

void calcDesiredTurn(void)
{
 // calculate where we need to turn to head to destination
 headingError = targetHeading - currentHeading;

 // adjust for compass wrap
 if (headingError < -180)
        headingError += 360;
 if (headingError > 180)
        headingError -= 360;
 // calculate which turn direction to display on LED's
#ifdef LED
 if (abs(headingError) <= HEADING_TOLERANCE)      // if within tolerance, indicate we
are going approx. straight
 {
        //turnDirection = straight;
        digitalWrite(straight, HIGH);
        digitalWrite(left, LOW);
        digitalWrite(right, LOW);
 }
 else if (headingError < 0)
 {
        //turnDirection = left;
        digitalWrite(straight, LOW);
        digitalWrite(left, HIGH);
        digitalWrite(right, LOW);
 }
 else if (headingError > 0)
 {
        //turnDirection = right;
        digitalWrite(straight, LOW);
        digitalWrite(left, LOW);
        digitalWrite(right, HIGH);
 }
 else
 {
        //turnDirection = straight;
        digitalWrite(straight, HIGH);
        digitalWrite(left, LOW);
```

```
        digitalWrite(right, LOW);
 }
#endif


}
// calcDesiredTurn()

//Calculates distance from current location to waypoint and increments waypoint # if
waypoint is reached
float distanceToWaypoint()
{

 float delta = radians(currentLong - targetLong);
 float sdlong = sin(delta);
 float cdlong = cos(delta);
 float lat1 = radians(currentLat);
 float lat2 = radians(targetLat);
 float slat1 = sin(lat1);
 float clat1 = cos(lat1);
 float slat2 = sin(lat2);
 float clat2 = cos(lat2);
 delta = (clat1 * slat2) - (slat1 * clat2 * cdlong);
 delta = sq(delta);
 delta += sq(clat2 * sdlong);
 delta = sqrt(delta);
 float denom = (slat1 * slat2) + (clat1 * clat2 * cdlong);
 delta = atan2(delta, denom);
 distanceToTarget = delta * 6372795;

 // check to see if we have reached the current waypoint
 if (distanceToTarget <= WAYPOINT_DIST_TOLERANCE)
 {
        nextWaypoint();
 }

 return distanceToTarget;
}  // distanceToWaypoint()

// Calculates heading to the waypoint from the current locaiton
// returns course in degrees (North=0, West=270) from position 1 to position 2,
// both specified as signed decimal-degrees latitude and longitude.
int courseToWaypoint()
{
 float dlon = radians(targetLong - currentLong);
 float cLat = radians(currentLat);
 float tLat = radians(targetLat);
 float a1 = sin(dlon) * cos(tLat);
 float a2 = sin(cLat) * cos(tLat) * cos(dlon);
 a2 = cos(cLat) * sin(tLat) - a2;
 a2 = atan2(a1, a2);
 if (a2 < 0.0)
 {
        a2 += TWO_PI;
 }
 targetHeading = degrees(a2);
 return targetHeading;
```

```
}   // courseToWaypoint()

// Called after new GPS data is received; updates our position and course/distance to
waypoint
void processGPS(void)
{
 currentLat = convertDegMinToDecDeg(GPS.latitude);
 currentLong = convertDegMinToDecDeg(GPS.longitude);

 if (GPS.lat == 'S')            // make them signed
        currentLat = -currentLat;
 if (GPS.lon = 'W')
        currentLong = -currentLong;

 // update the course and distance to waypoint based on our new position
 distanceToWaypoint();
 courseToWaypoint();

}   // processGPS(void)

// converts lat/long from Adafruit degree-minute format to decimal-degrees; requires
<math.h> library
double convertDegMinToDecDeg(float degMin)
{
 double min = 0.0;
 double decDeg = 0.0;

 //get the minutes, fmod() requires double
 min = fmod((double)degMin, 100.0);

 //rebuild coordinates in decimal degrees
 degMin = (int)(degMin / 100);
 decDeg = degMin + (min / 60);

 return decDeg;
}

//increment to next waypoint
void nextWaypoint(void)
{
 waypointNumber++;
 targetLat = waypointList[waypointNumber].getLat();
 targetLong = waypointList[waypointNumber].getLong();
#ifdef DEBUG
 Serial.print("Waypoint #: ");
 Serial.println(waypointNumber);
 Serial.println("targetLat:");
 Serial.println(targetLat, 6);
 Serial.println("targetLong");
 Serial.println(targetLong, 6);
#endif

 if ((targetLat == 0 && targetLong == 0) || waypointNumber >= NUMBER_WAYPOINTS)    //
last waypoint reached?
 {
#if defined(DEBUG) || defined(LCD)
#ifdef LCD
```

```
        Serial.write(12); //clear
        delay(5);
#endif
        Serial.print(F("Waypoint #: "));
        Serial.println(waypointNumber);
        Serial.write(148);
        Serial.println(F("*  LAST WAYPOINT!  *"));
        Serial.write(168);
        Serial.println(F("Joseph Varikooty"));
        Serial.write(188);
        Serial.print(F("Ryan Wallace"));
#endif
        digitalWrite(BRAKE_Right, HIGH);  //brake motors
        digitalWrite(BRAKE_Left, HIGH);   //brake motors
#if !defined(LED)
        loopForever();
#endif
#ifdef LED
        while (1)      //flash led to indicate that we've reached last waypoint
        {
               digitalWrite(end, HIGH);
               delay(100);
               digitalWrite(end, LOW);
               delay(100);
        }
#endif
 }

 processGPS();
 distanceToTarget = originalDistanceToTarget = distanceToWaypoint();
 courseToWaypoint();

}  // nextWaypoint()

//PD control
int PDcontrol(int headingError)
{
 float HEADING_KP = 0.9;             //Proportional gain
 float HEADING_KD = 0.00; //.2;             //Derivative gain
 int derror = ((headingError - last_error) / 2);
 int ierror = ((headingError + last_error) / 2);
 float diff = (HEADING_KP * headingError) + (HEADING_KD * derror);
 PD_output = diff;
 last_error = headingError;
 motorCorrection = map(PD_output, -150, 150, -maxCorrection, maxCorrection);
 return motorCorrection;
}

//controls motor to move
void move(int motorCorrection)
{
 digitalWrite(BRAKE_Right, LOW);  // setting brake LOW disable motor brake
 digitalWrite(DIR_Right, LOW);    // setting direction to HIGH the motor will spin
forward
 digitalWrite(BRAKE_Left, LOW);   // setting brake LOW disable motor brake
 digitalWrite(DIR_Left, LOW);     // setting direction to HIGH the motor will spin
forward
```

```
 motorspeedL = (baselineMotorSpeed + motorCorrection);     //motor speed corrected for
heading
 motorspeedR = (baselineMotorSpeed - motorCorrection);     //motor speed corrected for
heading
 motorspeedL = map(motorspeedL, 0, 255, 0, motorLEFTmaxFOWARD);    //motor speed
corrected for asymmetrical motor characteristics
 motorspeedR = map(motorspeedR, 0, 255, 0, motorRIGHTmaxFOWARD);   //motor speed
corrected for asymmetrical motor characteristics
#ifdef Move
 analogWrite(PWM_Left, motorspeedL);     // Set the speed of the motor
 analogWrite(PWM_Right, motorspeedR);    // Set the speed of the motor
#endif
}//move()

//print LCD
#ifdef LCD
void printLCD(void)
{
 Serial.write(12);            //clear screen
 Serial.write(17);            // turn on backlight
 delay(5);
 Serial.print(F("Waypt:"));
 Serial.println((waypointNumber));
 Serial.write(136);
 Serial.print(F("of"));
 Serial.println(NUMBER_WAYPOINTS);
 Serial.write(141);
 Serial.print("x=");
 Serial.print(distanceToWaypoint());
 Serial.write(148);
 Serial.print(F("TRG:"));
 Serial.print(targetHeading, DEC);
 Serial.write(156);
 Serial.print(F("HDG:"));
 Serial.print(currentHeading, DEC);
 Serial.write(163);
 //Serial.println(headingError, DEC);
 Serial.print(F("|S#"));
 //Serial.print(F("S"));
 Serial.print((int)GPS.satellites);
 //Serial.print(F("Q"));
 //Serial.println((int)GPS.fixquality);
 Serial.write(168);
 Serial.print(F("LAT="));
 Serial.print(currentLat, 5);
 Serial.write(188);
 Serial.print(F("LON="));
 Serial.println(currentLong, 5);
 Serial.write(181);
 Serial.print(F("|LM:"));
 Serial.println(motorspeedL);
 Serial.write(201);
 Serial.print(F("|RM:"));
 Serial.println(motorspeedR);
}
#endif
```

```
//print debugging data
#ifdef DEBUG
void printdebug(void)
{
 Serial.print("GPS Fix:");
 Serial.println((int)GPS.fix);
 Serial.println("Number of Satellites");
 Serial.println((int)GPS.satellites);
 Serial.print(" Fix Quality: ");
 Serial.println((int)GPS.fixquality);
 Serial.print(F("LAT = "));
 Serial.print(currentLat, 6);
 Serial.print(F(" LON = "));
 Serial.println(currentLong, 6);
 Serial.print("Waypoint #: ");
 Serial.println(waypointNumber);
 Serial.print("Waypoint LAT =");
 Serial.print(waypointList[waypointNumber].getLat(), 6);
 Serial.print(F(" Long = "));
 Serial.print(waypointList[waypointNumber].getLong(), 6);
 Serial.print(F(" Dist "));
 Serial.print(distanceToWaypoint());
 Serial.print(F(" Original Dist "));
 Serial.println(originalDistanceToTarget);
 Serial.print(F("Compass Heading "));
 Serial.println(currentHeading);
 Serial.print(F("GPS Heading "));
 Serial.println(GPS.angle);
 Serial.print(F("  Target = "));
 Serial.print(targetHeading, DEC);
 Serial.print(F("  Current = "));
 Serial.print(currentHeading, DEC);
 Serial.print(F("  Error = "));
 Serial.println(headingError, DEC);
 Serial.print("PD_Output: ");
 Serial.println(PD_output);
 Serial.print("Motor Correction: ");
 Serial.println(motorCorrection);
 Serial.print("Left Motor Speed: ");
 Serial.println(motorspeedL);
 Serial.print("Right Motor Speed: ");
 Serial.println(motorspeedR);
 Serial.print(F("Free Memory: "));
 Serial.println(freeRam(), DEC);
 Serial.print("Time: ");
 Serial.println(millis() - start_time);
}       //printdebug()
#endif
```