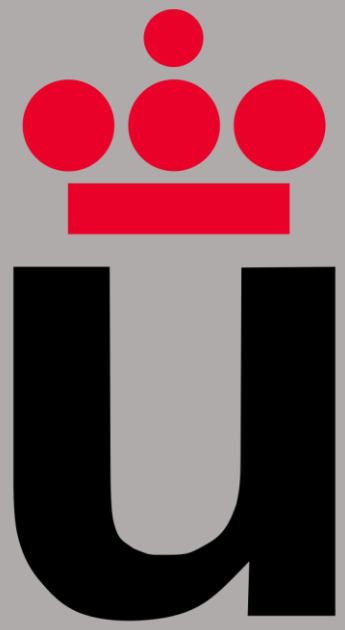
MEMORIA CARRITO GPS



Pedro
Barquero
Torres
Grupo 11

Contenido

1-Objetivos

El objetivo del proyecto es diseñar un carro que por bluetooth se conecta al móvil y a través de un módulo GPS y una brújula magnética sigue los pasos de la persona que lo está dirigiendo por el móvil.

Este carro sería una herramienta muy útil para evitar la carga de peso para las personas (bolsas, escombros de obras, neveras, etc).

2-Parte Hardware

En cuanto a la parte hardware hablare de los materiales, presupuesto y utilidad para el funcionamiento del proyecto. Para ello nos apoyaremos en el esquema de conexiones.

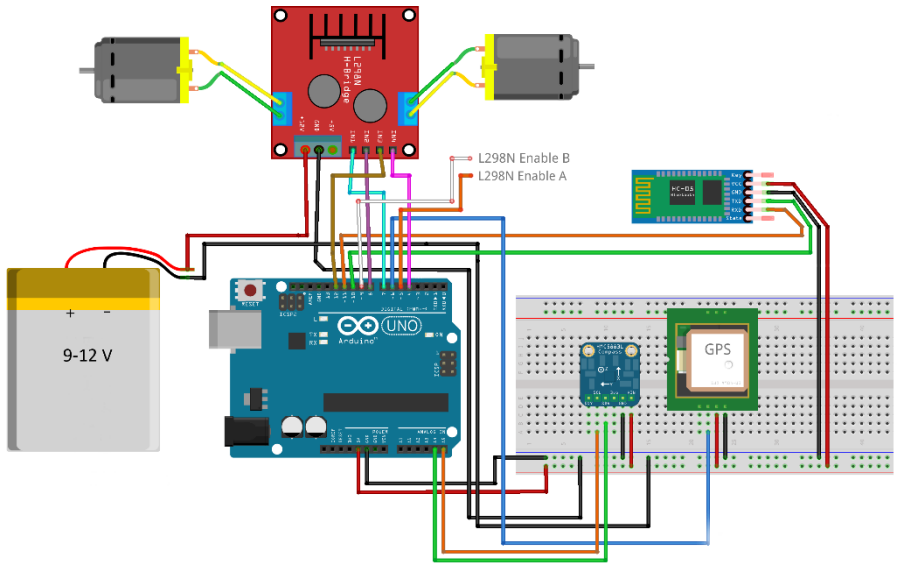
2.1 – Materiales y presupuesto

Los materiales utilizados para el proyecto son los siguientes:

MATERIAL	PRECIO
Placa Arduino UNO	URJC
Módulo GPS	9,70€
Brújula magnética	8,90€
Motor 12 V x 2	6€
Módulo Bluetooth	7€
Motor Driver	7,99€
Cajón de madera	Propio
Cables jumper	1,20€
Batería 12V	3,20€
Ruedas	Propio
Total	43,99

2.2 – Componentes y conexiones

Tras conectar los diferentes módulos a la placa de Arduino y a la batería, obtuve el siguiente esquema de conexiones:



En este esquema podemos distinguir los siguientes componentes:

Módulo bluetooth HC-05

El módulo bluetooth permite la conexión entre un teléfono y nuestro Arduino, para ello utilizaremos una aplicación que nos sirva de intermediario. En este caso será Blynk.

El módulo bluetooth tiene 6 pines, de los cuales únicamente haremos uso de 4: VCC (conectado a la fuente de 5V), GND (conectado al GND digital), RX (conectado al pin digital 11) y TX (conectado al pin digital 10).

Módulo GPS NEO-6M-0-001

Su función será la de enviar las coordenadas GPS del carro al Arduino para que este pueda compararlas con las coordenadas del móvil y así poder mover el carro hacia las coordenadas marcadas por el móvil.

Está formado por 5 pines, de los cuales únicamente haremos uso de 3: VCC (conectado a la fuente de 5V), GND (conectado al GND digital) y TX (conectado al pin digital 6).

No haremos uso del pin de RX ya que no recibirá ninguna información, solo transmitirá.

Brújula Magnética HMC5883

Es un componente fundamental para el funcionamiento del proyecto, ya que el carro debe saber dónde se encuentran las distintas coordenadas polares con la finalidad de poder seguir las coordenadas GPS enviadas por el móvil.

Posee 5 pines de los cuales haremos uso de 4: VCC (conectado a la fuente de 5V), GND (conectado al GND digital), SDA (conectado al pin analógico A4) y SCL (conectado al pin analógico A5).

Módulo L298N

Este módulo permite el control de los motores conectados a las 4 ranuras que posee en sus laterales. Podremos controlarlos mediante los 6 pines de entrada al Arduino. Este módulo tiene otras 3 ranuras usadas para darle alimentación a los motores y al propio módulo. Por lo que este módulo es alimentado directamente mediante la batería y no por el Arduino.

De los 6 pines de entradas 2 se utilizan para configurar la velocidad de los motores: Enable A (conectado al pin digital 5) y Enable B (conectado al pin digital 9). Los otros 4 pines se utilizan para programar la rotación de los motores: IN1(conectado al pin digital 7), IN2(conectado al pin digital 8), IN3(conectado al pin digital 12) y IN4(conectado al pin digital 4). Con esto IN1 e IN2, servirán para el control de un motor, IN3 e IN4 para el control del otro.

En las 4 ranuras que tiene en sus laterales conectaremos los polos positivos y negativos de los motores (1 motor por cada lado). Da igual en que orden conectarlos ya que se puede modificar el control del giro por software.

Y para terminar en las otras 3 ranuras conectaremos por un lado la fuente de 12V (batería) y por el otro el GND que será un GND combinado del GND digital del Arduino y el polo negativo de la batería.

Batería de 12 V

Es necesaria una batería de 12 V ya que sino los motores no tienen la suficiente fuerza para mover el mecanismo.

3-Parte Software

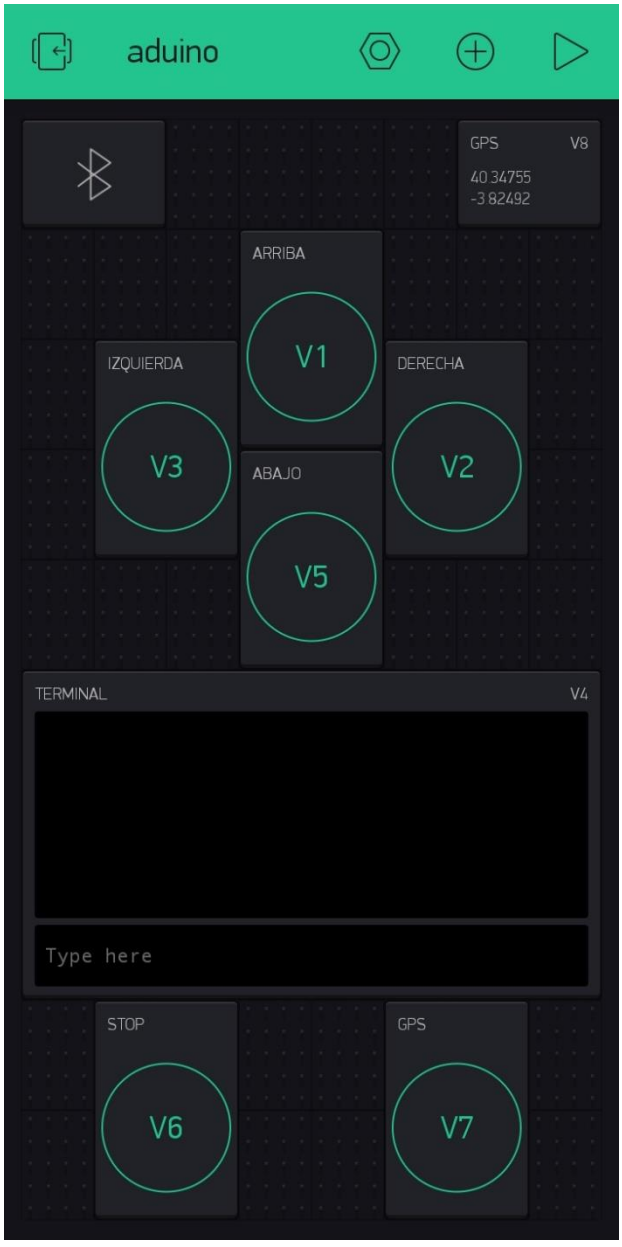
Debido a que el funcionamiento del proyecto depende del uso del móvil, la parte software se dividirá por un lado en el código utilizado para el Arduino y por otro lado en la aplicación que nos sirve como intermediario en el teléfono Android.

3.1- Blynk

Blynk se trata de una plataforma que permite controlar proyectos de Arduino mediante la creación de una interfaz gráfica de usuario.

Esta aplicación utiliza un sistema de pines digitales (V0-V31) a los cuales podremos acceder desde nuestro programa de Arduino para modificar las diferentes funcionalidades.

Para nuestro proyecto será la siguiente interfaz:



Donde los pines virtuales V1, V2, V3, V5 y V6 sirven para mover el carro mediante bluetooth (hacia delante, derecha, izquierda, hacia atrás y parar respectivamente). El pin virtual V8, sirve para enviarle al Arduino las coordenadas GPS del móvil (se realizará de manera automática cada cierto tiempo) y el pin V7, activa la función de seguir al usuario que posea el móvil. Por último, existe un terminal conectado al pin virtual V4 que nos irá explicando lo que está sucediendo en nuestro Arduino.

Todo esto podemos hacerlo introduciendo la librería especifica de la aplicación:

#include <BlynkSimpleSerialBLE.h>

3.2 – Código Arduino

3.2.1- Definición y configuración

Utilizaremos las siguientes librerías:

```
//librerias
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>
#include <Servo.h>
#include <SoftwareSerial.h>
#include <BlynkSimpleSerialBLE.h>
#include "./TinyGPS.h"
```

Y definiremos las siguientes constantes:

```
//Definicion
char auth[] = "B1rGOPW6QUwpx5Np1b-RF0oCb9_rwBuK";
#define BLUETOOTH_TX_PIN 10
#define BLUETOOTH_RX_PIN 11
#define GPS_TX_PIN 6
#define MOTOR_A_EN_PIN 5
#define MOTOR_B_EN_PIN 9
#define MOTOR_A_IN_1_PIN 7
#define MOTOR_A_IN_2_PIN 8
#define MOTOR_B_IN_1_PIN 12
#define MOTOR_B_IN_2_PIN 4
```

Que hacen referencia a los pines en los cuales tenemos conectados los distintos componentes.

```
//Brujula
#define DECLINATION_ANGLE -0.003f //eror del campo magnetico
#define COMPASS_OFFSET 0.0f
```

```
//GPS
#define GPS_UPDATE_INTERVAL 1000
#define GPS_STREAM_TIMEOUT 18
#define GPS_WAYPOINT_TIMEOUT 45
```

Elementos necesarios para el control de la brújula y del GPS.

Con esto realizamos las siguientes configuraciones para establecer la conexión en serie:

```
//bluetooth
SoftwareSerial bluetoothSerial(BLUETOOTH_TX_PIN, BLUETOOTH_RX_PIN);
SoftwareSerial nss(GPS_TX_PIN, 255);
```

También se define la siguiente estructura:

```
struct GeoLoc {
  float lat;
  float lon;
};
```

Que se corresponde con la longitud y la latitud de una determinada coordenada GPS.

Y se definen y declaran las siguientes variables:

```
Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);
TinyGPS gps;

////Variables de control
bool arriba = false ;
bool abajo = false ;
bool izquierda = false ;
bool derecha = false;
bool stp = false ; //stop
WidgetTerminal terminal(V4);

bool seguir = false;
```

Nuestro setup () sería el siguiente:

```
void setup()
{
   //Terminal movil
   terminal.clear();

  // Brujula
  setupCompass();

  //Motor
   pinMode(MOTOR_A_IN_1_PIN, OUTPUT);
  pinMode(MOTOR_A_IN_2_PIN, OUTPUT);
  pinMode(MOTOR_B_IN_1_PIN, OUTPUT);
  pinMode(MOTOR_B_IN_2_PIN, OUTPUT);
   pinMode(MOTOR_A_EN_PIN, OUTPUT);
  pinMode(MOTOR_B_EN_PIN, OUTPUT);

   //GPS
  nss.begin(9600);

 Serial.begin(9600);

      //Bluetooth
  bluetoothSerial.begin(9600);
  Blynk.begin(bluetoothSerial, auth);



}
```

En el cual, establecemos los pines como salidas, comenzamos las comunicaciones en serie ,limpiamos el terminal del móvil y configuramos la brújula.

Y el loop() :

```
void loop()
{
     Blynk.run();
}
```

Va a depender de la aplicación con la que enlazamos el Arduino (Blynk).

3.2.2 – Casos de uso

3.2.2.1-Modo bluetooth:

```
//*********************************************************
// CONTROL POR BLUETOOTH
//*********************************************************
BLYNK_WRITE(V1) {

 Serial.println("Modo bluetooth adelante ") ;
 terminal.println("Modo bluetooth adelante ") ;
 terminal.flush();
 bool arriba = true;
bool abajo = false ;
bool izquierda = false ;
bool derecha = false;
bool stp = false;
run(arriba,abajo,izquierda,derecha,stp);
}
BLYNK_WRITE(V2) {
  Serial.println("Modo bluetooth derecha ") ;
   terminal.println("Modo bluetooth derecha ") ;
   terminal.flush();

  bool arriba = false ;
bool abajo = false ;
bool izquierda = false ;
bool derecha = true;
bool stp = false;
run(arriba,abajo,izquierda,derecha,stp);
}
BLYNK_WRITE(V3) {
  Serial.println("Modo bluetooth izquierda ") ;
    terminal.println("Modo bluetooth izquierda ") ;
     terminal.flush();
     bool arriba = false ;
bool abajo = false ;
bool izquierda = true;
bool derecha = false;
bool stp = false;
run(arriba,abajo,izquierda,derecha,stp);

}BLYNK_WRITE(V5) {
  Serial.println("Modo bluetooth atras ") ;
   terminal.println("Modo bluetooth atras ") ;
```

```
    terminal.flush();
   bool arriba = false ;
bool abajo = true ;
bool izquierda = false;
bool derecha = false;
bool stp = false;
run(arriba,abajo,izquierda,derecha,stp);
}
BLYNK_WRITE(V6) {

  Serial.println("Modo bluetooth parar ") ;
   terminal.println("Modo bluetooth parar ") ;
    terminal.flush();
   bool arriba = false ;
bool abajo = false;
bool izquierda = false;
bool derecha = false;
bool stp = true;
run(arriba,abajo,izquierda,derecha,stp);
}
void run(bool arriba, bool abajo, bool izquierda,bool derecha, bool stp){
   analogWrite(MOTOR_A_EN_PIN,255);
    analogWrite(MOTOR_B_EN_PIN,255);

    Serial.println(arriba);
    Serial.println(abajo);
    Serial.println(izquierda);
    Serial.println(derecha);
     Serial.println(stp);

  if(arriba){
  digitalWrite(MOTOR_A_IN_1_PIN, HIGH);
  digitalWrite(MOTOR_A_IN_2_PIN, LOW);
  digitalWrite(MOTOR_B_IN_1_PIN, HIGH);
  digitalWrite(MOTOR_B_IN_2_PIN, LOW);
  }
  if(abajo){
  digitalWrite(MOTOR_A_IN_1_PIN, LOW);
  digitalWrite(MOTOR_A_IN_2_PIN, HIGH);
  digitalWrite(MOTOR_B_IN_1_PIN, LOW);
  digitalWrite(MOTOR_B_IN_2_PIN, HIGH);
    }
     if(derecha){
  digitalWrite(MOTOR_A_IN_1_PIN, LOW);
  digitalWrite(MOTOR_A_IN_2_PIN, LOW);
  digitalWrite(MOTOR_B_IN_1_PIN, HIGH);
  digitalWrite(MOTOR_B_IN_2_PIN, LOW);
    }
     if(izquierda){
  digitalWrite(MOTOR_A_IN_1_PIN, HIGH);
  digitalWrite(MOTOR_A_IN_2_PIN, LOW);
  digitalWrite(MOTOR_B_IN_1_PIN, LOW);
  digitalWrite(MOTOR_B_IN_2_PIN, LOW);
    }
     if(stp){
  digitalWrite(MOTOR_A_IN_1_PIN, LOW);
  digitalWrite(MOTOR_A_IN_2_PIN, LOW);
  digitalWrite(MOTOR_B_IN_1_PIN, LOW);
  digitalWrite(MOTOR_B_IN_2_PIN, LOW);
    }
  delay(1000);
  }
 //*****************************************************************
 //*****************************************************************
 //*********************************TERMINAL ANDROID**********
BLYNK_WRITE(V4) {
  terminal.println("Hola mundo");
  terminal.flush();

  }
```

Utilizamos las variables booleanas arriba, abajo, izquierda, derecha y stp para saber en que dirección queremos mover el carro o incluso si queremos pararlo.

Según el botón (o pin virtual) que apretemos en nuestro Android estas variables cambiaran su valor y se las pasaremos a la función *'run'* que es la que se encargara de mover el carro cambiando el valor de los pines asociados a los motores a *'HIGH'* o a *'LOW'* según corresponda.

3.2.2.2Modo GPS

```
//Control para que nos siga el carro
BLYNK_WRITE(V7) {
  seguir = !seguir;
  stop();

}
//Control para proporcionale al carro las coordenadas gps del movil
BLYNK_WRITE(V8) {
  GpsParam gps(param);

  Serial.println("Recibidas coordenadas GPS android : ");
  terminal.println("Recibidas coordenadas GPS android : ");


  Serial.print(gps.getLat(), 7); Serial.print(", "); Serial.println(gps.getLon(), 7);
   terminal.print(gps.getLat(), 7); terminal.print(", "); terminal.println(gps.getLon(), 7);
     terminal.flush();

  GeoLoc phoneLoc;
  phoneLoc.lat = gps.getLat();
  phoneLoc.lon = gps.getLon();

  driveTo(phoneLoc, GPS_STREAM_TIMEOUT);
}
```

Con estas funciones configuramos el uso de los pines virtuales V7 y V8 para que uno al pulsarlo active la función para que el carro nos siga (V7) y el otro para que cada cierto tiempo nos mande al Arduino la señal GPS del móvil (V8).

Además, crearemos las siguientes funciones de movimiento:

```
//*************************************************************************
//*****************************FUNCIONES DE MOVIMIENTO*********************************
void stop() {
  // now turn off motors
  digitalWrite(MOTOR_A_IN_1_PIN, LOW);
  digitalWrite(MOTOR_A_IN_2_PIN, LOW);
  digitalWrite(MOTOR_B_IN_1_PIN, LOW);
  digitalWrite(MOTOR_B_IN_2_PIN, LOW);

}
```

```
void drive(int distance, float turn) {


  int fullSpeed = 230;
  int stopSpeed = 0;

  // drive to location
  int s = fullSpeed;
  if ( distance < 8 ) {
    int wouldBeSpeed = s - stopSpeed;
    wouldBeSpeed *= distance / 8.0f;
    s = stopSpeed + wouldBeSpeed;
  }

  int autoThrottle = constrain(s, stopSpeed, fullSpeed);
  autoThrottle = 230;

  float t = turn;
  while (t < -180) t += 360;
  while (t >  180) t -= 360;

  Serial.print("turn: ");
  Serial.println(t);
  Serial.print("original: ");
  Serial.println(turn);

  float t_modifier = (180.0 - abs(t)) / 180.0;
  float autoSteerA = 1;
  float autoSteerB = 1;

  if (t < 0) {
    autoSteerB = t_modifier;
  } else if (t > 0){
    autoSteerA = t_modifier;
  }

  Serial.print("steerA: "); Serial.println(autoSteerA);
  Serial.print("steerB: "); Serial.println(autoSteerB);

  int speedA = (int) (((float) autoThrottle) * autoSteerA);
  int speedB = (int) (((float) autoThrottle) * autoSteerB);

  setSpeedMotorA(speedA);
  setSpeedMotorB(speedB);
}
```

```
void driveTo(struct GeoLoc &loc, int timeout) {
  nss.listen();
  GeoLoc carroLoc = checkGPS();
  bluetoothSerial.listen();

  if (carroLoc.lat != 0 && carroLoc.lon != 0 && seguir) {
    float d = 0;
    //Start move loop here
    do {
      nss.listen();
      carroLoc = checkGPS();
      bluetoothSerial.listen();

      d = geoDistance(carroLoc, loc);
      float t = geoBearing(carroLoc, loc) - geoHeading();

      Serial.print("Distancia : ");
      Serial.println(geoDistance(carroLoc, loc));

      Serial.print("rumbo: ");
      Serial.println(geoBearing(carroLoc, loc));

      Serial.print("grados: ");
      Serial.println(geoHeading());

      drive(d, t);
      timeout -= 1;
    } while (d > 3.0 && seguir && timeout>0);

    stop();
  }
}
```

La primera función sirve para parar el carro, la segunda para ponerlo en movimiento y determinar la velocidad de los motores (para poder girar) y por último la tercera calcula la distancia que el carro debe recorrer y lo pone en marcha con el objetivo de alcanzar las coordenadas GPS del Android.

Además de estas funciones se utilizan otras funciones de la librería TinyGPS y Adafruit_HMC5883 como son:

```
GeoLoc checkGPS() {
 Serial.println("Leyendo GPS carro: ");
 terminal.println("Leyendo GPS carro : \n");
 terminal.flush();
 bool newdata = false;
 unsigned long start = millis();
 while (millis() - start < GPS_UPDATE_INTERVAL) {
   if (feedgps())
     newdata = true;
 }
 if (newdata) {
   return gpsdump(gps);
 }

 GeoLoc carroLoc;
 carroLoc.lat = 0.0;
carroLoc.lon = 0.0;

 return carroLoc;
}

// En caso de tener datos nuevos los procesaremos :
GeoLoc gpsdump(TinyGPS &gps) {
 float flat, flon;
 unsigned long age;

 gps.f_get_position(&flat, &flon, &age);

 GeoLoc carroLoc;
 carroLoc.lat = flat;
 carroLoc.lon = flon;

 Serial.print(carroLoc.lat, 7); Serial.print(", "); Serial.println(carroLoc.lon, 7);
 terminal.print(carroLoc.lat, 7); terminal.print(", "); terminal.println(carroLoc.lon, 7);
 terminal.print("Conseguidas las coordenadas del carro ") ;
   terminal.flush();
 return carroLoc;
}
```

```
// proporciona datos a medida que estan listos
bool feedgps() {
  while (nss.available()) {
    if (gps.encode(nss.read()))
      return true;
  }
  return false;
}
```

```
float geoBearing(struct GeoLoc &a, struct GeoLoc &b) {
  float y = sin(b.lon-a.lon) * cos(b.lat);
  float x = cos(a.lat)*sin(b.lat) - sin(a.lat)*cos(b.lat)*cos(b.lon-a.lon);
  return atan2(y, x) * RADTODEG;
}

float geoDistance(struct GeoLoc &a, struct GeoLoc &b) {
  const float R = 6371000; // km
  float p1 = a.lat * DEGTORAD;
  float p2 = b.lat * DEGTORAD;
  float dp = (b.lat-a.lat) * DEGTORAD;
  float dl = (b.lon-a.lon) * DEGTORAD;

  float x = sin(dp/2) * sin(dp/2) + cos(p1) * cos(p2) * sin(dl/2) * sin(dl/2);
  float y = 2 * atan2(sqrt(x), sqrt(1-x));

  return R * y;
}

float geoHeading() {
  sensors_event_t event;
  mag.getEvent(&event);
  float heading = atan2(event.magnetic.y, event.magnetic.x);
  heading -= DECLINATION_ANGLE;
  heading -= COMPASS_OFFSET;
  if(heading < 0)
    heading += 2*PI;
  if(heading > 2*PI)
    heading -= 2*PI;
  float headingDegrees = heading * 180/M_PI;
  while (headingDegrees < -180) headingDegrees += 360;
  while (headingDegrees >  180) headingDegrees -= 360;
  return headingDegrees;
}
```

```
void setSpeedMotorA(int speed) {
  digitalWrite(MOTOR_A_IN_1_PIN, HIGH);
  digitalWrite(MOTOR_A_IN_2_PIN, LOW);
  analogWrite(MOTOR_A_EN_PIN, speed + MOTOR_A_OFFSET);
}

void setSpeedMotorB(int speed) {
  digitalWrite(MOTOR_B_IN_1_PIN, HIGH);
  digitalWrite(MOTOR_B_IN_2_PIN, LOW);
  analogWrite(MOTOR_B_EN_PIN, speed + MOTOR_B_OFFSET);
}

void setSpeed(int speed)
{
  setSpeedMotorA(speed);
  setSpeedMotorB(speed);
}
```

4- Problemas y posibles soluciones

El mayor problema encontrado es que los motores de 12V no alcanzan la fuerza suficiente como para mover la estructura del carro, por lo que se necesitarían motores con mayor número de RPM.

Otro problema es que el GPS del carro tarda mucho tiempo en hacer conexión con un satélite por lo que se hace muy tedioso probar el funcionamiento del código. Posible solución, utilizar un GPS con mayor precisión.

5-Bibliografía

https://www.arduino.cc/reference/en/libraries/tinygps/

http://arduiniana.org/libraries/tinygpsplus/

https://www.arduino.cc/reference/en/libraries/adafruit-hmc5883-unified/

https://github.com/adafruit/Adafruit_HMC5883_Unified/blob/master/Adafruit_HMC5883_U.cpp

https://github.com/adafruit/Adafruit_HMC5883_Unified/blob/master/Adafruit_HMC5883_U.h

https://github.com/mikalhart/TinyGPS/blob/master/TinyGPS.h

https://github.com/mikalhart/TinyGPS/blob/master/TinyGPS.cpp