Quoc Anh Tran
12/11/2021

CURRENT PROBLEMS:

1. **The battery dies out very fast (<30mins running) after being fully charged**
2. **IMU sensor is interfered with the metal chassis causing wrong reading on the North.**

A01 and B01 is positive
A02 and B02 is negative
Looking at the regulator L1086 CT - 5: GND is the left pin, OUTPUT is the middle pin, INPUT is the right pin
In Arduino:

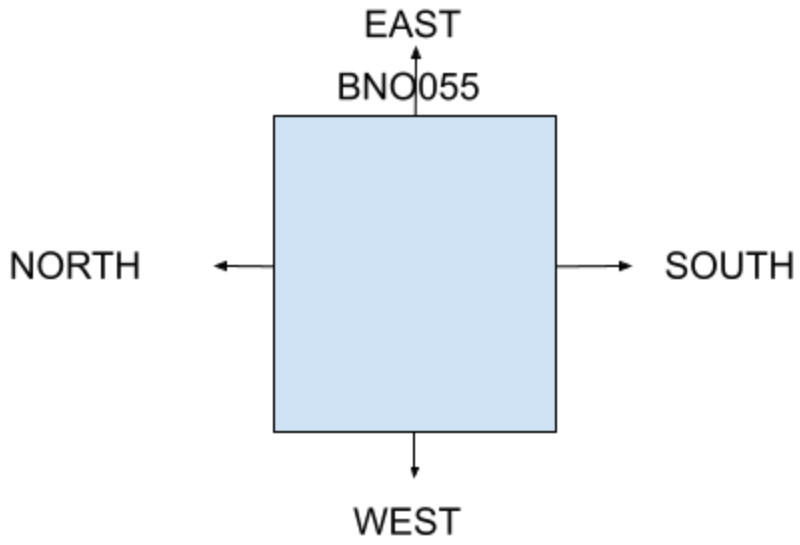
1. Flexi timer doesn't work with distance function to use Lidar
2. Flexi timer can’t process long instruction correctly such as lidar, wheels,... but it works with servo
3. Timer interrupts only work with servo with timer0 and timer2 as they are 8-bit timers. Other timers that are 16 bit or 32 bit, don't work with the servo library

Only PWM signal (either A or B) needs PWM pins in the Arduino board

Forward and Backward function must need a delay right after the use
of the function.

IMU sensor:

- **East is 0 degree, North is 270degree (+-10 degree error)**
- **The direction of the IMU name is East. This IMU reads North on the long edge to the left of its name.**



Bearing would be measured from North direction i.e 0° bearing means North, 90° bearing is East, 180° bearing is measured to be South, and 270° to be West.
https://www.igismap.com/formula-to-find-bearing-or-heading-angle-between-two-points-latitude-longitude/

Going Forward: The strength of the 2 wheels on the right is 8.5% higher than the strength of the 2 wheels on the left when they are set to the same speed. That means the car is going to the left at the same speed setting for 4 wheels. Tested distance: 10 meters

```
//forward(motor1, motor2, 109);
//forward(motor3, motor4, 120);
//delay(4000);
```

Going Backward: The strength of the 2 wheels on the right is 18%% higher than the strength of the 2 wheels on the left when they are set to the same speed. That means the car is going to the right at the same speed setting for 4 wheels. Tested distance: 10 meters

```
//back(motor3, motor4, 120);
//back(motor1, motor2, 98);
//delay(4000);
```

Turn left:

```
left(motor2,motor4,motor1,motor3,180,199);
delay(1000);
```

Standardize one 360 degrees turn for the speed of 128. That means I fix the speed to be 128/255.

```
right(motor2,motor4,motor1,motor3,128,128);
```

delay(7200); //It takes 7.2sec to finish about one circle (360 degrees) at 128/255 speed.

X degree turn = $\frac{X*7.2}{360}$

90 degrees turn = $\frac{90*7.2}{360}$ = 1.8sec. It takes 1.8 seconds to turn 90 degrees.

180 degrees turn = $\frac{180*7.2}{360}$ = 3.6sec. It takes 3.6 seconds to turn 180 degrees.

Coding formula = Angle*7200/360;
https://forum.arduino.cc/t/arduino-mega-2560-interrupt-issue-ctc-mode/553831/7
https://moji1.github.io/EEE499/labssrc/lab2/lab2.html
https://www.instructables.com/Arduino-Timer-Interrupts/
https://nerd-corner.com/arduino-timer-interrupts-how-to-program-arduino-registers/
https://www.robotshop.com/community/forum/t/arduino-101-timers-and-interrupts/13072
https://playground.arduino.cc/Code/Timer1/

Flexi library:
https://github.com/wimleers/flexitimer2

Function in TinyGPS library:

1. //lat/long in MILLIONTHs of a degree and age of fix in milliseconds
   void **get_position**(long *latitude, long *longitude, unsigned long *fix_age = 0);
2. // date as ddmmyy, time as hhmmsscc, and age in milliseconds
   void **get_datetime**(unsigned long *date, unsigned long *time, unsigned long *age = 0);
3. // signed altitude in centimeters (from GPGGA sentence)
   inline long **altitude**() { return _altitude; }
4. // course in last full GPRMC sentence in 100th of a degree
   inline unsigned long **course**() { return _course; }
5. // speed in last full GPRMC sentence in 100ths of a knot
   inline unsigned long **speed**() { return _speed; }
6. // satellites used in last full GPGGA sentence
   inline unsigned short **satellites**() { return _numsats; }
7. // horizontal dilution of precision in 100ths
   inline unsigned long **hdop**() { return _hdop; }
8. void **f_get_position**(float *latitude, float *longitude, unsigned long *fix_age = 0);
9. void **crack_datetime**(int *year, byte *month, byte *day, byte *hour, byte *minute, byte *second, byte *hundredths = 0, unsigned long *fix_age = 0);
10. float f_altitude();
11. float f_course();
12. float f_speed_knots();

```
    13.  float f_speed_mph();
    14.  float f_speed_mps();
    15.  float f_speed_kmph();
    16.
```

```
  static float distance_between (float lat1, float long1, float lat2, float long2);
  static float course_to (float lat1, float long1, float lat2, float long2);
  static const char *cardinal(float course);

float TinyGPS::course_to (float lat1, float long1, float lat2, float long2)
{
  // returns course in degrees (North=0, West=270) from position 1 to position 2,
  // both specified as signed decimal-degrees latitude and longitude.
  // Because Earth is no exact sphere, calculated course may be off by a tiny fraction.
  // Courtesy of Maarten Lamers
  float dlon = radians(long2-long1);
  lat1 = radians(lat1);
  lat2 = radians(lat2);
  float a1 = sin(dlon) * cos(lat2);
  float a2 = sin(lat1) * cos(lat2) * cos(dlon);
  a2 = cos(lat1) * sin(lat2) - a2;
  a2 = atan2(a1, a2);
  if (a2 < 0.0)
  {
    a2 += TWO_PI;
  }
  return degrees(a2);
}
```

```
//GPS_IMU_WHEELS CONTROL
/**************************************************************************
**************************************************************************
*******************************
*  Adafruit BNO055 Sensor Calibration sample
*  Refer to http://www.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf
*  Section 3.10 on how to calibrate your sensor:
*   -Run this sketch
*   -To calibrate the Gyroscope, just let the sensor sit for a couple of seconds until GYR
Calibration shows 1
*   -To calibrate the Magnetometer, move your sensor in random figure 8 patterns until MAG
Calibration shows 1
*   -If you want to calibrate the acceleration sensor, move the sensor to 6 stable positions slowly,
3 of those should be in the XY/XZ/YZ plane
*   -Once you are happy with the readings you get, copy the c_data = .... line from serial.console
to the setup() of your sketch and write it to the sensor with "bno.setCalibData(c_data);"
**************************************************************************
**************************************************************************
*******************************/

#include <Adafruit_BNO055.h>
#include <Adafruit_Sensor.h>
#include <utility/imumaths.h>
#include <Wire.h>
#include <Servo.h>
#include <LIDARLite.h>
#include <SparkFun_TB6612.h>

//Connect with pin 18(RX) and 19(TX) for TinyGPS
#include <TinyGPS.h>

// Globals

// first two motors
#define AIN1 2
#define BIN1 7
#define AIN2 4
#define BIN2 8
#define PWMA 5
#define PWMB 6
```

```
#define STBY 9

//second two motors
#define AIN12 13
#define BIN12 10
#define AIN22 12
#define BIN22 14
#define PWMA2 11
#define PWMB2 3
#define STBY2 15

//#define PI 3.14159265
// these constants are used to allow you to make your motor configuration
// line up with function names like forward.  Value can be 1 or -1
const int offsetA = 1;
const int offsetB = 1;

// Initializing motors.  The library will allow you to initialize as many
// motors as you have memory for.  If you are using functions like forward
// that take 2 motors as arguements you can either write new functions or
// call the function more than once.
Motor motor1 = Motor(AIN1, AIN2, PWMA, offsetA, STBY); //top right wheel
Motor motor2 = Motor(BIN1, BIN2, PWMB, offsetB, STBY); //top left wheel

Motor motor3 = Motor(AIN12, AIN22, PWMA2, offsetA, STBY2); //Bottom right wheel
Motor motor4 = Motor(BIN12, BIN22, PWMB2, offsetB, STBY2); //bottom left wheel


Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

//Global variables

//long   lat,lon; // create variable for latitude and longitude object
float lat,lon;
float distan = 0;
TinyGPS gps; // create gps object
Adafruit_BNO055 bno = Adafruit_BNO055(55);      // define a BNO055 object

void setup(){
```

```
Serial.begin(9600); // connect serial
Serial.println("The GPS Received Signal:");
Serial1.begin(9600); // connect gps sensor
Serial.println("Orientation Sensor Calibration"); Serial.println("");

  /* Initialise the sensor */
  if (!bno.begin(Adafruit_BNO055::OPERATION_MODE_NDOF)) //if you want to calibrate using another mode, set it here.
  {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");     // if the sensor is not found!
    while (1);
  }

  // You will need to put the next two lines into your own sketch, _immediatly_ after bno.begin() to use a predefined calibration
  // byte c_data[22] = {0, 0, 0, 0, 0, 0, 172, 250, 112, 255, 52, 253, 0, 0, 253, 255, 255, 255, 232, 3, 240, 2}; //replace this line with the serial output of this sketch

  byte c_data[22] = {31, 0, 237, 255, 234, 255, 49, 2, 137, 1, 36, 255, 0, 0, 0, 0, 2, 0, 232, 3, 67, 3};
  bno.setCalibData(c_data);

  delay(1000);
  bno.setExtCrystalUse(true);
}

void loop(){

  float bearingAngle = 0;
  float headingAngle = 0;
  float turningAngle = 0;
  sensors_event_t event;
  bno.getEvent(&event);

  //print the euler angles for reference
  //Serial.print("X: "); //"heading"
  //Serial.println(headingAngle, 4);
```

```
/*Serial.print(" Y: ");
Serial.print(event.orientation.y, 4);
Serial.print(" Z: ");
Serial.print(event.orientation.z, 4);
Serial.println();*/
  if(distan > 4){
     forward(motor1, motor2, 109);
     forward(motor3, motor4, 120);
     delay(100);
     Serial.println("distan > 4m");
   }
 while(Serial1.available()){ // check for gps data

  if(gps.encode(Serial1.read()))// encode gps data
  {
  gps.f_get_position(&lat,&lon); // get latitude and longitude

  Serial.print("Position: ");

  //Latitude
  Serial.print("Latitude: ");
  Serial.print(lat,6);

  Serial.print(",");

  //Longitude
  Serial.print("Longitude: ");
  Serial.println(lon,6);

  distan = gps.distance_between(lat,lon,37.976256,-121.320794);
  Serial.print("Distance: ");
  Serial.println(distan,6);

  bearingAngle = gps.course_to(lat,lon,37.976256,-121.320794);//bearing angle
  Serial.print("Bearing angle to destination: ");
  Serial.println(bearingAngle,2);

  headingAngle = event.orientation.x;
  Serial.print("X: "); //"heading"
  Serial.println(headingAngle, 4);
```

```
  delay(100);

  turningAngle = bearingAngle - headingAngle;
  Serial.print("turningAngle: ");
  Serial.println(turningAngle, 4);
  if(distan > 4){
    if(turningAngle < -180){
      //turn right
      right(motor2,motor4,motor1,motor3,128,128);
      delay((turningAngle+360)*7200/360);
      Serial.println("Right: <-180");
    }
    else if(turningAngle >= -180 && turningAngle < -5){
      //turn left
      left(motor2,motor4,motor1,motor3,128,128);
      delay(fabs(turningAngle)*7200/360);
      Serial.println("left < -5");
    }
    else if(turningAngle > 5 && turningAngle <= 180){
      //turn right
      right(motor2,motor4,motor1,motor3,128,128);
      delay((turningAngle)*7200/360);
      Serial.println("right <=180");
    }
    else if(turningAngle > 180){
      //turn left
      left(motor2,motor4,motor1,motor3,128,128);
      delay((360-turningAngle)*7200/360);
      Serial.println("left > 180");
    }
    else{
        forward(motor1, motor2, 109);
        forward(motor3, motor4, 120);
        delay(100);
        Serial.println("last going forward");
    }
   }

  }
}
```

}

```
//Obstacle Avoidance
#include <FlexiTimer2.h>
#include <Wire.h>
#include <Servo.h>
#include <LIDARLite.h>
#include <SparkFun_TB6612.h>

// Globals

// first two motors
#define AIN1 2
#define BIN1 7
#define AIN2 4
#define BIN2 8
#define PWMA 5
#define PWMB 6
#define STBY 9

//second two motors
#define AIN12 13
#define BIN12 10
#define AIN22 12
#define BIN22 14
#define PWMA2 11
#define PWMB2 3
#define STBY2 15

//#define PI 3.14159265
// these constants are used to allow you to make your motor configuration
// line up with function names like forward.  Value can be 1 or -1
const int offsetA = 1;
const int offsetB = 1;

// Initializing motors.  The library will allow you to initialize as many
// motors as you have memory for.  If you are using functions like forward
// that take 2 motors as arguements you can either write new functions or
// call the function more than once.
Motor motor1 = Motor(AIN1, AIN2, PWMA, offsetA, STBY); //top right wheel
Motor motor2 = Motor(BIN1, BIN2, PWMB, offsetB, STBY); //top left wheel
```

```
Motor motor3 = Motor(AIN12, AIN22, PWMA2, offsetA, STBY2); //Bottom right wheel
Motor motor4 = Motor(BIN12, BIN22, PWMB2, offsetB, STBY2); //bottom left wheel


Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

//Global variables
int pos = 60;    // variable to store the servo position
int i = 0;
LIDARLite lidarLite;
int angleArray[3] = {0,0,0};
float distArray[20] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

float convertFromDegreeToRadian(int degree){
  return (degree*PI/180);
}
float returnVerticalDistance(float distance, float objectAngle){
  objectAngle = convertFromDegreeToRadian(objectAngle);
  return distance*sin(objectAngle);
}

void flash()
{
  int count = 0;
  int dist;
  float dV;
  /*while(i == 3){
    brake(motor1, motor2);
    brake(motor3, motor4);
    delay(100);
    i = 0;
    for (pos = 60; pos <= 120; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    dist = lidarLite.distance(); // With bias correction
    dV = returnVerticalDistance(dist,pos);
    if(dV < 70){
      if(i < 3){
```

```
      angleArray[i] = pos;
      if( i > 0 && ((angleArray[i] - 1) != angleArray[i-1])){
        i = 0;
      }
      else {  i++; }
     }
     if((i == 3) && a < 20){
      distArray[a] = dV;        // even location gives distant
      distArray[a + 1] = angleArray[1];     //odd location gives angle at the previous even location
      i = 0;
      a += 2;
     }
    }

    //Serial.print(pos);
    //Serial.println(" : current angle");
    //Serial.print(dist);
    //Serial.print(" cm; dV = ");
    //Serial.println(dV);
    delay(1);                       // waits 15 ms for the servo to reach the position
   }

   float minDist = distArray[0];
   for(int j = 0; j <10; j++){
    if((distArray[2*j] != 0) && (distArray[2*j] <= minDist)){
     minDist = distArray[2*j];
     detectedAngle = distArray[2*j + 1];
    }
   }
}
    if(detectedAngle != 0){
     //Serial.print(bestAngle);
     //Serial.println(" : flash degree");
     if(detectedAngle <= 90){
      right(motor2,motor4,motor1,motor3,128,128);
      delay(detectedAngle*7200/360);
     }
     else{
      left(motor2,motor4,motor1,motor3,128,128);
      delay((detectedAngle-90)*7200/360);
```

```
      }

      //Serial.print(bestAngle);
      //Serial.println(" : updated 0 degree");
    }
    detectedAngle = 0;
    i = 0;
    a = 0;
    for(int j = 0; j <20; j++){
      distArray[j] = 0;
    }*/
}
void setup() {
  Serial.begin(9600); // Initialize serial connection to display distance readings
  lidarLite.begin(0, true); // Set configuration to default and I2C to 400 kHz
  lidarLite.configure(0); // Change this number to try out alternate configurations
  myservo.attach(22);  // attaches the servo on pin 22 to the servo object

  //FlexiTimer2::set(1000, 1/1000, flash); // call every 200 1ms "ticks"
  //FlexiTimer2::start();

}

void loop() {
  int dist;
  int a = 0;
  int detectedAngle = 0;
  float dV;
  forward(motor1, motor2, 109);
  forward(motor3, motor4, 120);
  delay(100);

  for (pos = 60; pos <= 120; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    dist = lidarLite.distance(); // With bias correction
    dV = returnVerticalDistance(dist,pos);
    //Serial.print(pos);
    //Serial.println(" : current angle");
    //Serial.print(dist);
```

```
   //Serial.print(" cm; dV = ");
   //Serial.println(dV);
   if(dV < 50){
    if(i < 3){
     angleArray[i] = pos;
     if( i > 0 && ((angleArray[i] - 1) != angleArray[i-1])){
      i = 0;
     }
     else {  i++; }
    }
     if((i == 3) && a < 20){
      distArray[a] = dV;        // even location gives distant
      distArray[a + 1] = angleArray[2];     //odd location gives angle at the previous even
location
      Serial.print(distArray[a]);
      Serial.print(" : current dist; previous angle: ");
      Serial.println(distArray[a + 1]);
      i = 0;
      a += 2;
     }
    }

   delay(1);                     // waits 15 ms for the servo to reach the position
  }

  float minDist = distArray[0];
  for(int j = 0; j <10; j++){
   if((distArray[2*j] != 0) && (distArray[2*j] <= minDist)){
    minDist = distArray[2*j];
    detectedAngle = distArray[2*j + 1];
    Serial.print(minDist);
    Serial.print(" : min dist; previous angle: ");
    Serial.println(detectedAngle);
   }
  }
  if(detectedAngle != 0){
   Serial.print(detectedAngle);
   Serial.println(" : detected degree");
   brake(motor1, motor2);
   brake(motor3, motor4);
```

```
    delay(500);
     if(detectedAngle <= 90){
       Serial.println("turn left");
       left(motor2,motor4,motor1,motor3,128,128);
       delay(detectedAngle*7200/360);
     }
     else{
       Serial.println("turn right");
       right(motor2,motor4,motor1,motor3,128,128);
       delay((detectedAngle-90)*7200/360);
     }

     //Serial.print(bestAngle);
     //Serial.println(" : updated 0 degree");
   }
   detectedAngle = 0;
   i = 0;
   a = 0;
   for(int j = 0; j <20; j++){
     distArray[j] = 0;
     Serial.println(j);
   }
  for (pos = 120; pos >= 60; pos -= 1) { // goes from 180 degrees to 0 degrees
   myservo.write(pos);              // tell servo to go to position in variable 'pos'
   dist = lidarLite.distance(); // With bias correction
   dV = returnVerticalDistance(dist,pos);
   delay(1);                       // waits 15 ms for the servo to reach the position
  }
}
```

```
//Obstacle avoidance and IMU and GPS and wheel
/**************************************************************************
**************************************************************************
*******************************
*  Adafruit BNO055 Sensor Calibration sample
*  Refer to http://www.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf
*  Section 3.10 on how to calibrate your sensor:
*   -Run this sketch
*   -To calibrate the Gyroscope, just let the sensor sit for a couple of seconds until GYR
Calibration shows 1
*   -To calibrate the Magnetometer, move your sensor in random figure 8 patterns until MAG
Calibration shows 1
*   -If you want to calibrate the acceleration sensor, move the sensor to 6 stable positions slowly,
3 of those should be in the XY/XZ/YZ plane
*   -Once you are happy with the readings you get, copy the c_data = .... line from serial.console
to the setup() of your sketch and write it to the sensor with "bno.setCalibData(c_data);"
**************************************************************************
**************************************************************************
*******************************/

#include <Adafruit_BNO055.h>
#include <Adafruit_Sensor.h>
#include <utility/imumaths.h>
#include <Wire.h>
#include <Servo.h>
#include <LIDARLite.h>
#include <SparkFun_TB6612.h>

//Connect with pin 18(RX) and 19(TX) for TinyGPS
#include <TinyGPS.h>

// Globals

// first two motors
#define AIN1 2
#define BIN1 7
#define AIN2 4
#define BIN2 8
#define PWMA 5
#define PWMB 6
```

```
#define STBY 9

//second two motors
#define AIN12 13
#define BIN12 10
#define AIN22 12
#define BIN22 14
#define PWMA2 11
#define PWMB2 3
#define STBY2 15

//#define PI 3.14159265
// these constants are used to allow you to make your motor configuration
// line up with function names like forward.  Value can be 1 or -1
const int offsetA = 1;
const int offsetB = 1;

// Initializing motors.  The library will allow you to initialize as many
// motors as you have memory for.  If you are using functions like forward
// that take 2 motors as arguements you can either write new functions or
// call the function more than once.
Motor motor1 = Motor(AIN1, AIN2, PWMA, offsetA, STBY); //top right wheel
Motor motor2 = Motor(BIN1, BIN2, PWMB, offsetB, STBY); //top left wheel

Motor motor3 = Motor(AIN12, AIN22, PWMA2, offsetA, STBY2); //Bottom right wheel
Motor motor4 = Motor(BIN12, BIN22, PWMB2, offsetB, STBY2); //bottom left wheel


Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

//Global variables

//long   lat,lon; // create variable for latitude and longitude object
float lat,lon;
float distan = 0;

TinyGPS gps; // create gps object
Adafruit_BNO055 bno = Adafruit_BNO055(55);      // define a BNO055 object
```

```
int pos = 60;    // variable to store the servo position
int i = 0;
LIDARLite lidarLite;
int angleArray[3] = {0,0,0};
float distArray[20] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

float convertFromDegreeToRadian(int degree){
  return (degree*PI/180);
}
float returnVerticalDistance(float distance, float objectAngle){
  objectAngle = convertFromDegreeToRadian(objectAngle);
  return distance*sin(objectAngle);
}

void setup(){
  Serial.begin(9600); // Initialize serial connection to display distance readings
  lidarLite.begin(0, true); // Set configuration to default and I2C to 400 kHz
  lidarLite.configure(0); // Change this number to try out alternate configurations
  myservo.attach(22);  // attaches the servo on pin 22 to the servo object

  Serial.begin(9600); // connect serial
  Serial.println("The GPS Received Signal:");
  Serial1.begin(9600); // connect gps sensor
  Serial.println("Orientation Sensor Calibration"); Serial.println("");

  /* Initialise the sensor */
  if (!bno.begin(Adafruit_BNO055::OPERATION_MODE_NDOF)) //if you want to calibrate
using another mode, set it here.
  {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");     // if the
sensor is not found!
    while (1);
  }

  // You will need to put the next two lines into your own sketch, _immediatly_ after bno.begin()
to use a predefined calibration
  // byte c_data[22] = {0, 0, 0, 0, 0, 0, 172, 250, 112, 255, 52, 253, 0, 0, 253, 255, 255, 255, 232,
3, 240, 2}; //replace this line with the serial output of this sketch
```

```
  byte c_data[22] = {31, 0, 237, 255, 234, 255, 49, 2, 137, 1, 36, 255, 0, 0, 0, 0, 2, 0, 232, 3, 67,
3};
  bno.setCalibData(c_data);

  delay(1000);
  bno.setExtCrystalUse(true);

}

void loop(){
  int dist;
  int a = 0;
  int detectedAngle = 0;
  float dV;


  //print the euler angles for reference
  //Serial.print("X: "); //"heading"
  //Serial.println(headingAngle, 4);
  /*Serial.print(" Y: ");
  Serial.print(event.orientation.y, 4);
  Serial.print(" Z: ");
  Serial.print(event.orientation.z, 4);
  Serial.println();*/
  float turningAngle = 0;
  float bearingAngle = 0;
  float headingAngle = 0;
  sensors_event_t event;
  bno.getEvent(&event);

  //while(Serial1.available());

    while(Serial1.available()){ // check for gps data

    if(gps.encode(Serial1.read()))// encode gps data
    {
    gps.f_get_position(&lat,&lon); // get latitude and longitude

    Serial.print("Position: ");
```

```
//Latitude
Serial.print("Latitude: ");
Serial.print(lat,6);

//Serial.print(",");

//Longitude
//Serial.print("Longitude: ");
//Serial.println(lon,6);

distan = gps.distance_between(lat,lon,37.976256,-121.320794);
//Serial.print("Distance: ");
//Serial.println(distan,6);

bearingAngle = gps.course_to(lat,lon,37.976256,-121.320794);//bearing angle
//Serial.print("Bearing angle to destination: ");
//Serial.println(bearingAngle,2);

headingAngle = event.orientation.x;
//Serial.print("X: "); //"heading"
//Serial.println(headingAngle, 4);
delay(100);

turningAngle = bearingAngle - headingAngle;
//Serial.print("turningAngle: ");
//Serial.println(turningAngle, 4);
   if(distan > 4){
 if(turningAngle < -180){
   //turn right
   right(motor2,motor4,motor1,motor3,128,128);
   delay((turningAngle+360)*7200/360);
   Serial.println("Right: <-180");
 }
 else if(turningAngle >= -180 && turningAngle < -5){
   //turn left
   left(motor2,motor4,motor1,motor3,128,128);
   delay(fabs(turningAngle)*7200/360);
   Serial.println("left < -5");
 }
 else if(turningAngle > 5 && turningAngle <= 180){
```

```
    //turn right
    right(motor2,motor4,motor1,motor3,128,128);
    delay((turningAngle)*7200/360);
    Serial.println("right <=180");
   }
   else if(turningAngle > 180){
    //turn left
    left(motor2,motor4,motor1,motor3,128,128);
    delay((360-turningAngle)*7200/360);
    Serial.println("left > 180");
   }
   else{
      forward(motor1, motor2, 109);
      forward(motor3, motor4, 120);
      delay(100);
      Serial.println("last going forward");
   }
  }
 }
}

 if(distan > 4){
    forward(motor1, motor2, 109);
    forward(motor3, motor4, 120);
    delay(100);
   for (pos = 60; pos <= 120; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    dist = lidarLite.distance(); // With bias correction
    dV = returnVerticalDistance(dist,pos);
    //Serial.print(pos);
    //Serial.println(" : current angle");
    //Serial.print(dist);
    //Serial.print(" cm; dV = ");
    //Serial.println(dV);
    if(dV < 50){
      if(i < 3){
        angleArray[i] = pos;
        if( i > 0 && ((angleArray[i] - 1) != angleArray[i-1])){
          i = 0;
```

```
        }
        else {  i++; }
       }
        if((i == 3) && a < 20){
          distArray[a] = dV;        // even location gives distant
          distArray[a + 1] = angleArray[2];     //odd location gives angle at the previous even
location
          Serial.print(distArray[a]);
          Serial.print(" : current dist; previous angle: ");
          Serial.println(distArray[a + 1]);
          i = 0;
          a += 2;
        }
       }

    delay(1);                     // waits 15 ms for the servo to reach the position
  }

  float minDist = distArray[0];
  for(int j = 0; j <10; j++){
   if((distArray[2*j] != 0) && (distArray[2*j] <= minDist)){
    minDist = distArray[2*j];
    detectedAngle = distArray[2*j + 1];
    Serial.print(minDist);
    Serial.print(" : min dist; previous angle: ");
    Serial.println(detectedAngle);
   }
  }
  if(detectedAngle != 0){
   Serial.print(detectedAngle);
   Serial.println(" : detected degree");
   brake(motor1, motor2);
   brake(motor3, motor4);
   delay(500);
    if(detectedAngle <= 90){
     Serial.println("turn left");
     left(motor2,motor4,motor1,motor3,128,128);
     delay(detectedAngle*7200/360);
    }
    else{
```

```
      Serial.println("turn right");
      right(motor2,motor4,motor1,motor3,128,128);
      delay((detectedAngle-90)*7200/360);
     }

     //Serial.print(bestAngle);
     //Serial.println(" : updated 0 degree");
   }
   detectedAngle = 0;
   i = 0;
   a = 0;
   for(int j = 0; j <20; j++){
     distArray[j] = 0;
     Serial.println(j);
   }
}

if(distan > 4){
 if(turningAngle < -180){
   //turn right
   right(motor2,motor4,motor1,motor3,128,128);
   delay((turningAngle+360)*7200/360);
   //Serial.println("Right: <-180");
 }
 else if(turningAngle >= -180 && turningAngle < -5){
   //turn left
   left(motor2,motor4,motor1,motor3,128,128);
   delay(fabs(turningAngle)*7200/360);
   //Serial.println("left < -5");
 }
 else if(turningAngle > 5 && turningAngle <= 180){
   //turn right
   right(motor2,motor4,motor1,motor3,128,128);
   delay((turningAngle)*7200/360);
   //Serial.println("right <=180");
 }
 else if(turningAngle > 180){
   //turn left
   left(motor2,motor4,motor1,motor3,128,128);
   delay((360-turningAngle)*7200/360);
```

```
      //Serial.println("left > 180");
    }
    else{
      forward(motor1, motor2, 109);
      forward(motor3, motor4, 120);
      delay(100);
      //Serial.println("last going forward");
    }
  }
}
```