ECE452C_Fall-2017_Proj-02_Team-13

Link: https://www.youtube.com/watch?v=H7KT8yi0h3g&feature=share

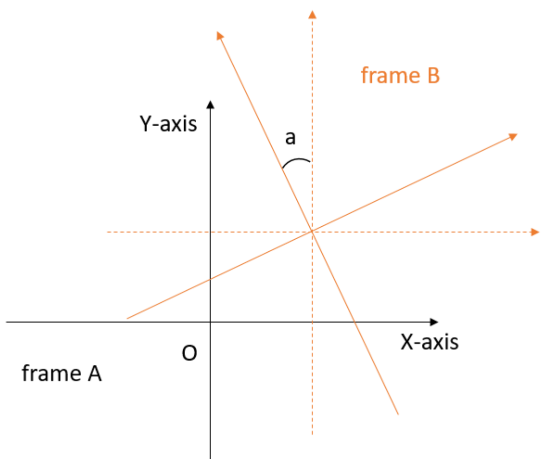
Group Member	Responsibility
Dong, Zhenglin	1. Finished writing bug0 and bug1; 2. Made the environment for the robot; 3. Tuning the parameter; 4. Helped finished the report of the project.
Pan, Hongyi	1. Finished writing bug0 and bug2; 2. Wrote important and basic functions which were used in the program; 3. Combined the code of geometric transformation and code of driving along the lines; 4. Made the video.
Zhang, Haopeng	1. Provided the key idea of how to write the algorithms specifically; 2. Helped correct several mistakes in bug0, bug1 and bug2. 3. Made the environment for the robot; 4. Made the video; 5. Finished the report of the project.

1. Principle thesis

It is a SE(2) problem.

Assume that A is ground frame, B is car frame and q is the goal point.

Step 1: Compute the initial q_B (q in car frame)



$q_B = g_{BA} * q_A$

$g_{AB} = g_{AB}^{-1}$

$$g_{AB} = \begin{matrix} cosa & -sina & x_b \\ sina & cosa & y_b \\ 0 & 0 & 1 \end{matrix}$$

(x_b and y_b are the car's coordinates in frame A. And "a" is the angle that frame B rotate around the Z-axis.)

Step 2: Compute q_B'

Assume in a very short time Δt, the right wheel and the left wheel drives for "rDistance" and "lDistance" respectively. Factor "d" is the distance between 2 wheels .

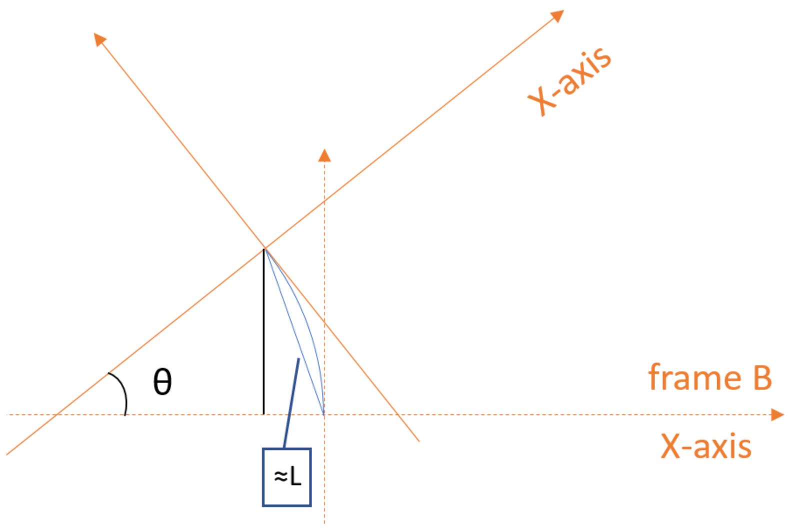
Based on our inferences, we can get the following data:

The angle the car rotated by: $\theta = \frac{\text{rDistance} - \text{lDistance}}{d}$

The distance the car drove: $L = \frac{\text{rDistance+lDistance}}{2}$

$$g_{BB'} = \begin{matrix} cos\theta & -sin\theta & -L * sin(\theta/2) \\ sin\theta & cos\theta & L * cos(\theta/2) \\ 0 & 0 & 1 \end{matrix}$$

$q_{B'} = g_{B'B} * q_B$



2. Procedure

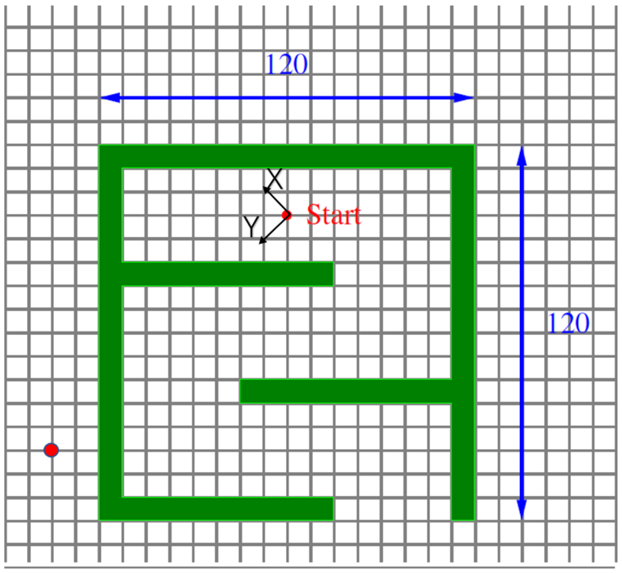
Bug 0 test:

#1: the goal configuration is (0, 1300) (mm), in our test, the robot can find the goal successfully.

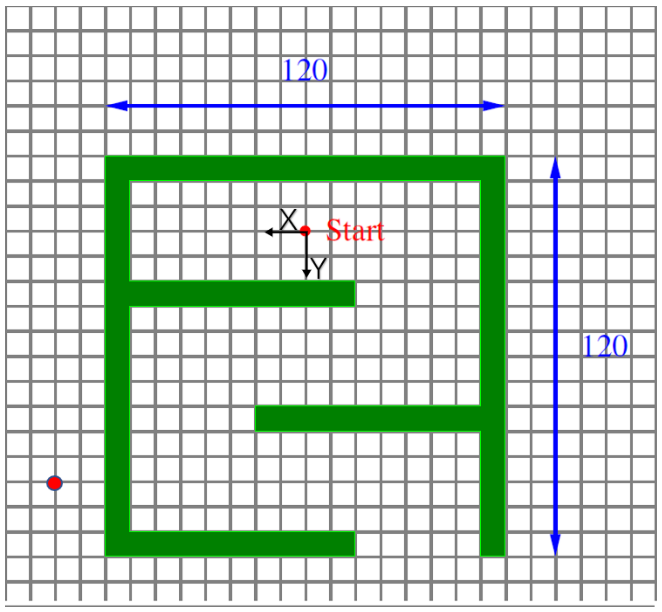
#2: the goal configuration is (-1000, 400) (mm), in our test, the robot cannot find the goal and it rotates in the right corner. Obviously, it accords with the problems of Bug 0 algorithm. Notice that in our video, the robot passes through the obstacles, because the first version of our map is not reasonable and the obstacles are too narrow.

We design another map properly and do the test 3 & 4.

#3: the goal configuration is (920, 920) (mm), in our test, the robot can find the goal successfully.



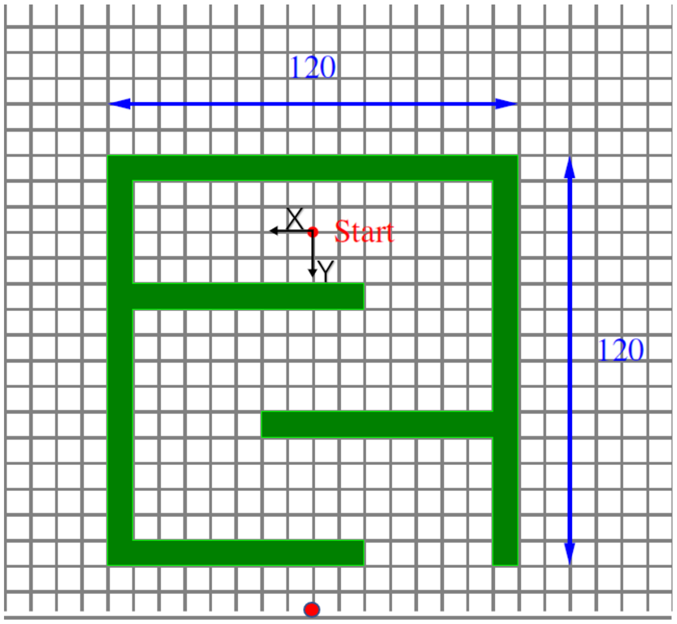
#4： the goal configuration is (0, 1300) (mm), in our test, the robot can find the goal successfully.



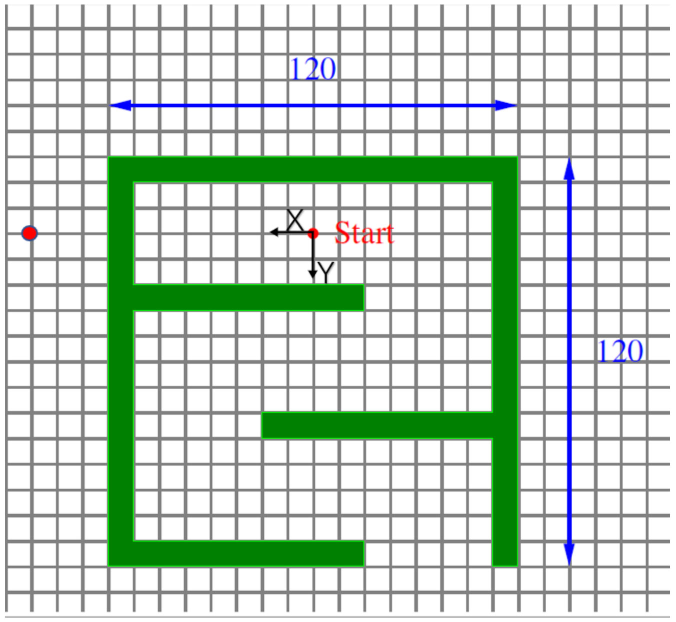
Overall, our Bug 0 algorithm is perfect. The car can totally find its way as we expected. However, bug0 naturally has flaws, which leads to failure in finding some certain goal points.

Bug 1 test:

#1: the goal configuration is (0, 1500) (mm). In out test, we tried millions of times to get a proper result.



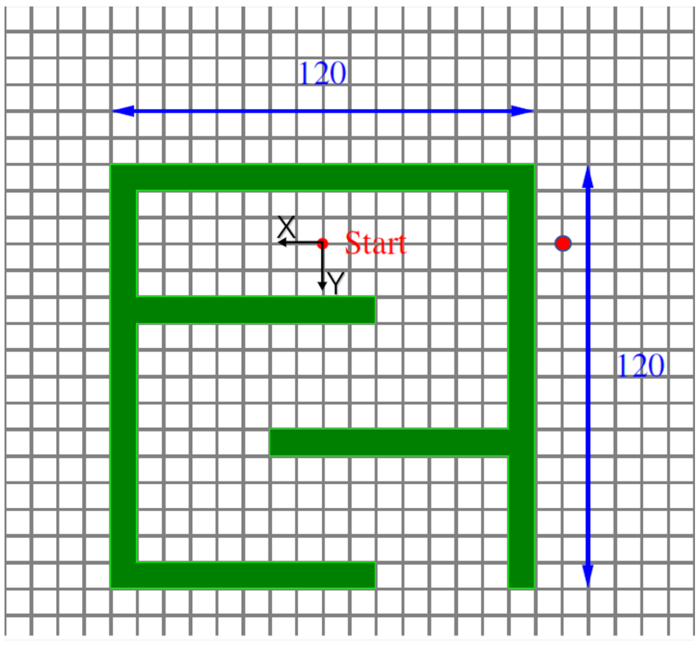
#2: the goal configuration is (1200, 0) (mm). As we can see in the video, the result is worse than the “bug1 test#1”. The car didn’t fully finish circumnavigating the obstacle. However, “bug1 test#2” is the second best result we can get due to several uncertain reasons.



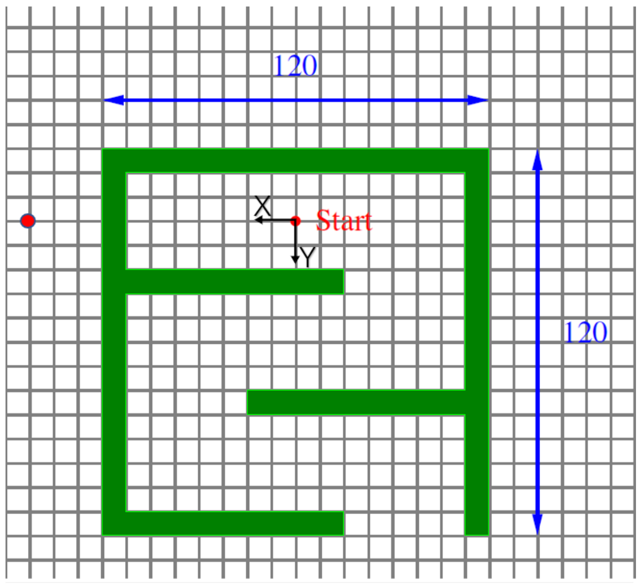
Overall, though we think our logics are totally our bug1 algorithm has several flaws itself. The configuration of the hit point and the goal point with respect to car frame cannot be computed accurately

<u>Bug 2 test</u>:

<u>#1</u>: the goal configuration is (-900,0) (mm).



<u>#2</u>: the goal configuration is (900, 0) (mm).



Overall, our bug2 tests ran into the same problems we met in bug1 tests. From some certain perspectives, this is reasonable because these two algorithms have many characteristics in common. For example, we compute hit point and goal point with respect to the car frame at the same time in bug1 as well as in bug2.

But in our project, the car drove shorter distance using bug2 than that using bug1. Maybe that's why the results of bug2 tests seems better than those of bug1 tests. However, the point still cannot be computed accurately.

3. Questions

(1) Is Bug 0 algorithm a good choice for this geometry? Why? Under which assumptions can Bug 0 find the goal?

The Bug 0 algorithm is not a good choice for this geometry. Because Bug 0 algorithm just simply rules the robot turn left when it meets the obstacles. In this geometry, there are many scenes in which the robot cannot find the goal.

As shown in the figure below, when we set the goal at the position on the right of the map, the robot cannot find a solution. And it turns circle in the corner marked in blue.

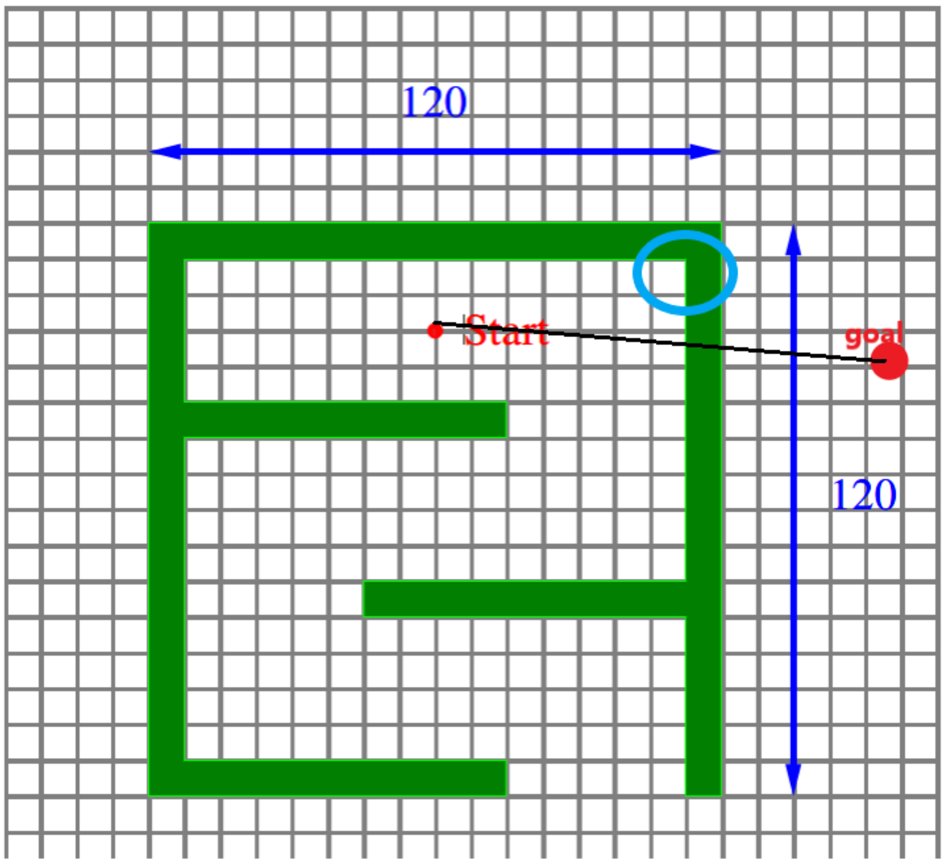
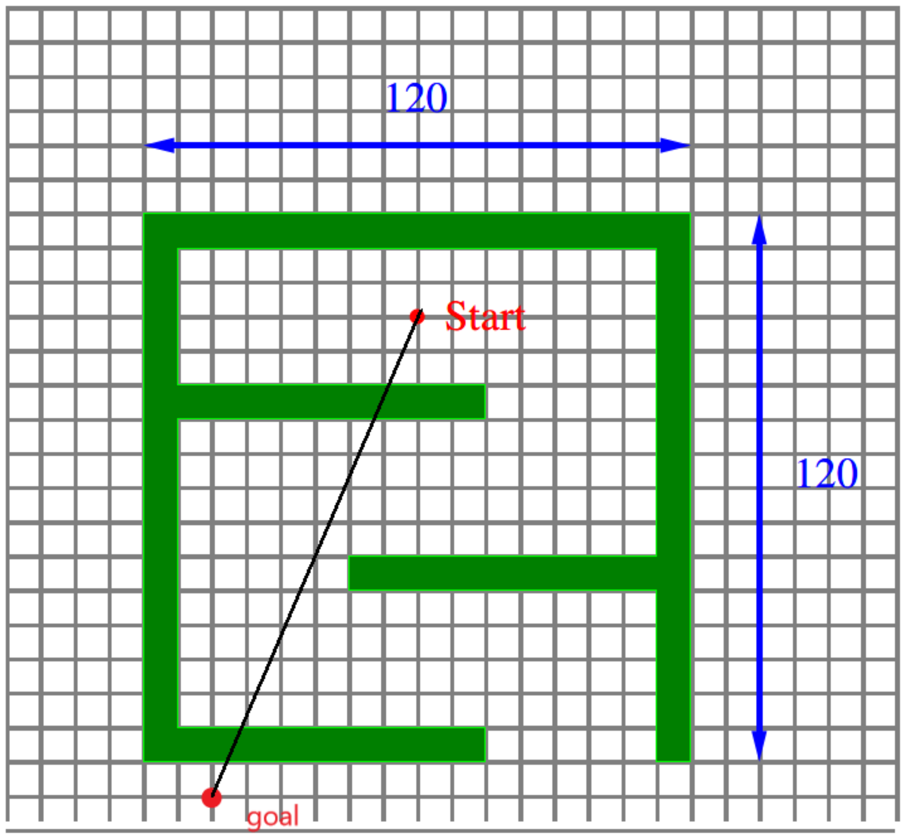


Figure 1: Robot environment

In some specific situations, Bug 0 can work and find the goal successfully. As shown in the figures below, in this situation the robot can find the solution and move to the goal.



(2) Is Bug 1 algorithm guaranteed to find the path to the goal in this geometry? Explain.

Bug 1 algorithm can definitely find the goal in this geometry.

The car can compute the distance from the goal while circumnavigating the obstacle. After circumnavigating the obstacle, it can circumvent the obstacle and move forward. In the end, it can circumvent all the obstacles between start point and the goal point. If the goal is not inside the obstacle, bug1 can always find the goal as well as bug2.

(3) Under which assumption can Bug 2 algorithm find the goal? Explain.

Under the assumption that the goal is not inside the obstacle.

The car cannot go through the obstacle to find the goal. Otherwise, bug2 can always find the goal in the end.

4. Difficulty #1:

1. 8 bit MCU cannot handle long type numbers well, thus sometimes there will be overflow or ridiculously enormous result.
2. Sensor is limited. We can only know how many round each wheel has rotated, but we cannot know exactly angle.
3. Map is too big to make. There are many rough edges where papers connect with others.

5. Solution #1:

1. Repeatedly test code. If the result is ridiculously enormous, try again.
2. Change sampling time.
3. When the robot is about to move to these edges, flat these places by pressing.

6. Code list: bug0, bug1 and bug2 are individual programs.

<u>Bug0</u>

```
#include <RedBot.h>
#include <math.h>
```

```
#define LED 13
#define BUTTON 12
#define LINETHRESHOLD 800
#define LEFTSPEED 60  // sets the nominal speed. Set to any number from 0 - 255.
#define RIGHTSPEED 60
#define countsPerRev 192
#define wheelDiam 65
#define wheelCirc (PI*wheelDiam)
#define delayTime 5
#define DELAYTIME 10
#define wheelDistance 160
#define OFFSET 1
RedBotSensor left = RedBotSensor(A3);   // initialize a left sensor object on A3
RedBotSensor center = RedBotSensor(A6); // initialize a center sensor object on A6
RedBotSensor right = RedBotSensor(A7);  // initialize a right sensor object on A7
RedBotMotors motors;
RedBotEncoder encoder = RedBotEncoder(A2, 10);
void get_q(double q[2], double x, double y);
void get_g(double g[3][3], double angle = 0, double x = 0, double y = 0);
void inv_g(double ginv[3][3], double g[3][3]);
void gq(double result[2], double g[3][3], double q[2]);
void driveLine(double * q);
void setup()
{
  // put your setup code here, to run once:
  int leftSpeed;   // variable used to store the leftMotor speed
  int rightSpeed;  // variable used to store the rightMotor speed
  long lCount, rCount, lDiff, rDiff, prevlCount, prevrCount;
  double lDistance, rDistance;
  double qA[2] = { 920, 920};
  double qB[2], gB1B2[3][3], gB2B1[3][3], gBA[3][3], gAB[3][3];
  double angle;
  Serial.begin(9600);
  pinMode(BUTTON, INPUT_PULLUP);
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);
  get_g(gAB, 0);//angle, x, y
  inv_g(gBA, gAB);
  gq(qB, gBA, qA);
  while(digitalRead(BUTTON) == HIGH);
  Serial.println(qB[0]);
  Serial.println(qB[1]);
  Serial.print("********\n");
  while(abs(qB[0])>50 || abs(qB[1])>50)//has reached?
```

```
 {
  if (qB[1]>0 && abs(atan(qB[0]/qB[1]))<0.08)//如果在正前方，则直行
  {
   // go straight
   leftSpeed = -LEFTSPEED;
   rightSpeed = RIGHTSPEED;
   motors.leftMotor(-leftSpeed);
   motors.rightMotor(-rightSpeed);
   encoder.clearEnc(BOTH);
   lDistance = 0;
   rDistance = 0;
   prevlCount = 0;
   prevrCount = 0;
//     delay(delayTime);
//     lCount = encoder.getTicks(LEFT);
//     rCount = encoder.getTicks(RIGHT);
//     lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
//     rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
//     qB[1] = sqrt(qB[1]*qB[1]+qB[0]*qB[0]) - (lDistance+rDistance)/2;
//     qB[0] = 0;

   while(left.read()<=LINETHRESHOLD && center.read()<=LINETHRESHOLD && right.read()<=LINETHRESHOLD)
   {
    delay(delayTime);
    lCount = encoder.getTicks(LEFT);
    rCount = encoder.getTicks(RIGHT);
    lCount = abs(lCount);
    rCount = abs(rCount);
    lDistance = double(lCount) * wheelCirc / countsPerRev;
    rDistance = double(rCount) * wheelCirc / countsPerRev;
    if (lDistance + rDistance >= 2*qB[1]) break;
    lDiff = (lCount - prevlCount);
    rDiff = (rCount - prevrCount);
    prevlCount = lCount;
    prevrCount = rCount;
    if (lDiff > rDiff) //has turned right, should turn left
    {
     leftSpeed += OFFSET;
     rightSpeed += OFFSET;
    }
    // if right is faster than the left, speed up the left / slow down right
    else if (lDiff < rDiff) //has turned left, should turn right
    {
      leftSpeed -= OFFSET;
      rightSpeed -= OFFSET;
    }
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
  }
  qB[1] -= (lDistance + rDistance)/2;
}
else
{
  if(qB[0] > 0)//如果在右，则右转
  {
    // turn right
    if(qB[1] == 0) angle = PI/2;
    else if(qB[1]>0) angle = atan(qB[0]/qB[1]);
    else angle = PI + atan(qB[0]/(qB[1]));
    leftSpeed = -LEFTSPEED;
    rightSpeed = -RIGHTSPEED;
    encoder.clearEnc(BOTH);
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
    do
    {
      delay(1);
      lCount = encoder.getTicks(LEFT);
      rCount = encoder.getTicks(RIGHT);
      lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
      rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
    }while (rDistance+lDistance < wheelDistance*angle);
  }
  else//否则左转
  {
    // turn left
    if(qB[1] == 0) angle = PI/2;
    else if(qB[1]>0) angle = atan(-qB[0]/qB[1]);
    else angle = PI + atan(-qB[0]/qB[1]);
    leftSpeed = LEFTSPEED;
    rightSpeed = RIGHTSPEED;
    encoder.clearEnc(BOTH);
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
    do
    {
      delay(1);
```

```
          lCount = encoder.getTicks(LEFT);
          rCount = encoder.getTicks(RIGHT);
          lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
          rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
        }while (rDistance+lDistance < wheelDistance*angle);
      }
      qB[1] = sqrt(qB[1]*qB[1]+qB[0]*qB[0]);
      qB[0] = 0;
    }
    /********** if hit an obstacle ***********/
    if(    (center.read()>    LINETHRESHOLD)    )         //||(left.read()>    LINETHRESHOLD)||(right.read()>
LINETHRESHOLD)
    {
      //turn left
//      digitalWrite(LED, HIGH);
      driveLine(qB);
//      digitalWrite(LED, LOW);
    }
  }
  motors.brake();
}
void loop()
{
  // put your main code here, to run repeatedly:
}
void get_q(double q[2], double x, double y)
{
  q[0] = x;
  q[1] = y;
}
void get_g(double g[3][3], double angle, double x, double y)
{
  g[0][0] = cos(angle);
  g[0][1] = -sin(angle);
  g[0][2] = x;
  g[1][0] = sin(angle);
  g[1][1] = cos(angle);
  g[1][2] = y;
  g[2][0] = 0;
  g[2][1] = 0;
  g[2][2] = 1;
}
void inv_g(double ginv[3][3], double g[3][3])
{
  double gg[3][3];
  unsigned char i, j;
  for(i=0; i<3; i++)
  {
    for(j = 0; j<3; j++)
    {
      gg[i][j] = g[i][j];
    }
  }
  ginv[0][0] = gg[0][0];
  ginv[0][1] = gg[1][0];
  ginv[0][2] = -gg[0][0]*gg[0][2] - gg[1][0]*gg[1][2];
  ginv[1][0] = gg[0][1];
  ginv[1][1] = gg[1][1];
  ginv[1][2] = -gg[0][1]*gg[0][2] - gg[1][1]*gg[1][2];
  ginv[2][0] = 0;
  ginv[2][1] = 0;
  ginv[2][2] = 1;
}
void gq(double result[2], double g[3][3], double q[2])
{
  double qq[2] = {q[0], q[1]};
  result[0] = g[0][2] + g[0][0]*qq[0] + g[0][1]*qq[1];
  result[1] = g[1][2] + g[1][0]*qq[0] + g[1][1]*qq[1];
}

void driveLine(double * q)
{

  int hit_state_now=0,hit_state_last=0;
  int count=0;
  int leftSpeed,rightSpeed;
  long lCount, rCount;
  double lDistance, rDistance;
  double g[3][3], g1[3][3];
  double theta;

  digitalWrite(LED,HIGH);
//  Serial.println(left.read());
//  Serial.println(center.read());
//  Serial.println(right.read());
//  Serial.println(q[0]);
//  Serial.println(q[1]);
//  Serial.print("============\n");
```

```
/************** turn left when hit an abstacle ****************/
//如果左传感器先 hit，那么需要向左转大弯
encoder.clearEnc(BOTH);
if( left.read()>LINETHRESHOLD&&center.read()<LINETHRESHOLD )
{

  while( count<2 )
  {

    leftSpeed = LEFTSPEED;
    rightSpeed = RIGHTSPEED;

    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);

    if( hit_state_now>hit_state_last )
    {
      count++;
    }
    hit_state_last=hit_state_now;
    if( center.read()<LINETHRESHOLD*0.9 )
      hit_state_now=0;
    if( center.read()>LINETHRESHOLD )
      hit_state_now=1;
   // Serial.print(hit_state_now);
  }
}

//正着进去，或者左斜着进去，不需要向左大转弯

leftSpeed = LEFTSPEED;
rightSpeed = RIGHTSPEED;
motors.leftMotor(-leftSpeed);
motors.rightMotor(-rightSpeed);

while(center.read() > LINETHRESHOLD);//
lCount = encoder.getTicks(LEFT);
rCount = encoder.getTicks(RIGHT);
lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;

get_g(g, (lDistance+rDistance)/wheelDistance);
inv_g(g1, g);
gq(q, g1, q);
```

```
//  Serial.println(q[0]);
//  Serial.println(q[1]);
//  Serial.print("=============\n");
//  motors.brake();
//  while(1);

  while(q[0] > 0 || q[1] < 0)
  {
   if(center.read() > LINETHRESHOLD) // if center sensor is above the line, go straight
   {
    leftSpeed = -LEFTSPEED;
    rightSpeed = RIGHTSPEED;
   }
   else if(left.read() > LINETHRESHOLD) // if left sensor is above the line, urn right
   {
    leftSpeed = -(LEFTSPEED - 50);
    rightSpeed = RIGHTSPEED + 50;
   }

   else if(right.read() > LINETHRESHOLD) // if right sensor is above the line, turn left
   {
    leftSpeed = -(LEFTSPEED + 50);
    rightSpeed = RIGHTSPEED - 50;
   }

   encoder.clearEnc(BOTH);     //编码器清零
   if(left.read()<LINETHRESHOLD || center.read()<LINETHRESHOLD || right.read()<LINETHRESHOLD)
   {
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
   }


   delay(DELAYTIME);
   lCount = encoder.getTicks(LEFT);
   rCount = encoder.getTicks(RIGHT);
   lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
   rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
   theta = (rDistance - lDistance) / wheelDistance;
   get_g(g, theta, -(lDistance+rDistance)*sin(theta/2)/2, (lDistance+rDistance)*cos(theta/2)/2);
   inv_g(g1, g);
   gq(q, g1, q);

  }
  digitalWrite(LED,LOW);
  Serial.println(q[0]);
  Serial.println(q[1]);
  Serial.print("============\n");
}
```

Bug1

```
#include <RedBot.h>
#include <math.h>
#define LED 13
#define BUTTON 12      // button to be pressed to start up
#define LINETHRESHOLD 800
#define LEFTSPEED 60  // sets the nominal speed. Set to any number from 0 - 255.
#define RIGHTSPEED 60
#define TURNSPEED 50
#define countsPerRev 192
#define wheelDiam 65
#define wheelCirc (PI*wheelDiam)
#define delayTime 5
#define DELAYTIME 10
#define wheelDistance 160
#define OFFSET 1
#define PHITTHRESHOLD 150  //permit error that might happen
RedBotSensor left = RedBotSensor(A3);   // initialize a left sensor object on A3
RedBotSensor center = RedBotSensor(A6); // initialize a center sensor object on A6
RedBotSensor right = RedBotSensor(A7);  // initialize a right sensor object on A7
RedBotMotors motors;
RedBotEncoder encoder = RedBotEncoder(A2, 10);
void get_q(double q[2], double x, double y);    // the function to get coordinates of the goal point in car
frame
void get_g(double g[3][3], double angle = 0, double x = 0, double y = 0);// the function to get the g as
gab
void inv_g(double ginv[3][3], double g[3][3]);// the function to get the g inverse: g1
void gq(double result[2], double g[3][3], double q[2]);// the function to get q' which equals g1*q
void driveLine(double * q); // the function to circumnavigate the obstacle in bug1
void driveDistance(double distance);// use project#1 algrithms to drive for a certain distance

void setup() {
  // put your setup code here, to run once:
  int leftSpeed;   // variable used to store the leftMotor speed
```

```
 int rightSpeed;  // variable used to store the rightMotor speed
 long lCount, rCount, lDiff, rDiff, prevlCount, prevrCount;
 double lDistance, rDistance;
 double qA[2] = { 0, 500};
 double qB[2], gB1B2[3][3], gB2B1[3][3], gBA[3][3], gAB[3][3];
 double angle;
 Serial.begin(9600);
 pinMode(BUTTON, INPUT_PULLUP);
 pinMode(LED, OUTPUT);
 digitalWrite(LED, LOW);
 get_g(gAB, 0);//angle, x, y
 inv_g(gBA, gAB);
 gq(qB, gBA, qA);
 while(digitalRead(BUTTON) == HIGH);
 Serial.println(qB[0]);
 Serial.println(qB[1]);
 Serial.print("********\n");
 while(abs(qB[0])>50 || abs(qB[1])>50)// while not at the goal
 {
  if (qB[1]>0 && abs(atan(qB[0]/qB[1]))<0.08)// if the goal is almost straight ahead
  {
   // then go straight
   leftSpeed = -LEFTSPEED;
   rightSpeed = RIGHTSPEED;
   motors.leftMotor(-leftSpeed);
   motors.rightMotor(-rightSpeed);
   encoder.clearEnc(BOTH);
   lDistance = 0;
   rDistance = 0;
   prevlCount = 0;
   prevrCount = 0;

   //while not hit obstacle, go strictly straight towards the goal
   while(left.read()<=LINETHRESHOLD && center.read()<=LINETHRESHOLD && right.read()<=LINETHRESHOLD)
   {
    delay(delayTime);
    lCount = encoder.getTicks(LEFT);
    rCount = encoder.getTicks(RIGHT);
    lCount = abs(lCount);
    rCount = abs(rCount);
    lDistance = double(lCount) * wheelCirc / countsPerRev;
    rDistance = double(rCount) * wheelCirc / countsPerRev;
    if (lDistance + rDistance >= 2*qB[1]) break;//if reaches the goal, break
    lDiff = (lCount - prevlCount);
    rDiff = (rCount - prevrCount);
    prevlCount = lCount;
    prevrCount = rCount;
    if (lDiff > rDiff) //has turned right, should turn left
    {
      leftSpeed += OFFSET;
      rightSpeed += OFFSET;
    }
    // if right is faster than the left, speed up the left / slow down right
    else if (lDiff < rDiff) //has turned left, should turn right
    {
      leftSpeed -= OFFSET;
      rightSpeed -= OFFSET;
    }
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
  }

  qB[1] -= (lDistance + rDistance)/2;//compute the coordinates of the goal q in car frame
}
else
{
  if(qB[0] > 0)//if the goal is on the right, then turn right
  {
    //compute the directon of the goal
    if(qB[1] == 0) angle = PI/2;
    else if(qB[1]>0) angle = atan(qB[0]/qB[1]);
    else angle = PI + atan(qB[0]/(qB[1]));
    // turn right until the car face towards the goal point q
    leftSpeed = -LEFTSPEED;
    rightSpeed = -RIGHTSPEED;
    encoder.clearEnc(BOTH);
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
    do
    {
      delay(1);
      lCount = encoder.getTicks(LEFT);
      rCount = encoder.getTicks(RIGHT);
      lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
      rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
    }while (rDistance+lDistance < wheelDistance*angle);
  }
```

```
      else  //otherwise, turn left
      {
        //compute the directon of the goal
        if(qB[1] == 0) angle = PI/2;
        else if(qB[1]>0) angle = atan(-qB[0]/qB[1]);
        else angle = PI + atan(-qB[0]/qB[1]);
        // turn left until the car face towards the goal point q
        leftSpeed = LEFTSPEED;
        rightSpeed = RIGHTSPEED;
        encoder.clearEnc(BOTH);
        motors.leftMotor(-leftSpeed);
        motors.rightMotor(-rightSpeed);
        do
        {
          delay(1);
          lCount = encoder.getTicks(LEFT);
          rCount = encoder.getTicks(RIGHT);
          lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
          rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
        }while (rDistance+lDistance < wheelDistance*angle);
      }

      //compute the goal point coordinates expressed in car frame
      qB[1] = sqrt(qB[1]*qB[1]+qB[0]*qB[0]);
      qB[0] = 0;
    }
    /********** if hit an obstacle ***********/
    if( (center.read()> LINETHRESHOLD) )
    {
      //turn left
      digitalWrite(LED, HIGH);
      driveLine(qB);
      digitalWrite(LED, LOW);
    }
  }
  motors.brake();//if reaches the goal, brake.
}


void loop() {
  // put your main code here, to run repeatedly:
}

/*********** the specific function of get_q get_g inv_g gq ************/
```

```
void get_q(double q[2], double x, double y)
{
  q[0] = x;
  q[1] = y;
}
void get_g(double g[3][3], double angle, double x, double y)
{
  g[0][0] = cos(angle);
  g[0][1] = -sin(angle);
  g[0][2] = x;
  g[1][0] = sin(angle);
  g[1][1] = cos(angle);
  g[1][2] = y;
  g[2][0] = 0;
  g[2][1] = 0;
  g[2][2] = 1;
}
void inv_g(double ginv[3][3], double g[3][3])
{
  double gg[3][3];
  unsigned char i, j;
  for(i=0; i<3; i++)
  {
    for(j = 0; j<3; j++)
    {
      gg[i][j] = g[i][j];
    }
  }
  ginv[0][0] = gg[0][0];
  ginv[0][1] = gg[1][0];
  ginv[0][2] = -gg[0][0]*gg[0][2] - gg[1][0]*gg[1][2];
  ginv[1][0] = gg[0][1];
  ginv[1][1] = gg[1][1];
  ginv[1][2] = -gg[0][1]*gg[0][2] - gg[1][1]*gg[1][2];
  ginv[2][0] = 0;
  ginv[2][1] = 0;
  ginv[2][2] = 1;
}
void gq(double result[2], double g[3][3], double q[2])
{
  double qq[2] = {q[0], q[1]};
  result[0] = g[0][2] + g[0][0]*qq[0] + g[0][1]*qq[1];
  result[1] = g[1][2] + g[1][0]*qq[0] + g[1][1]*qq[1];
}
```

```
/*********************** driveLine ************************/
void driveLine(double*q)
{
  int hit_state_now=0,hit_state_last=0;
  int count0=0;
  int leftSpeed,rightSpeed;
  long lCount, rCount;
  double lDistance, rDistance;
  double g[3][3], g1[3][3];
  double theta;
  double S,S1;
  double Dmin;
  double Phit[2]={ 0,0 };//hit point
  int count=0;
  /************** turn left when hit an abstacle ****************/
  //if the left sensor hits the obstacle first, then the car needs a big left turn
  encoder.clearEnc(BOTH);
  if( left.read()>LINETHRESHOLD&&center.read()<LINETHRESHOLD )//if the left sensor hits the obstacle
first
  {
    while( count0<2 )//if the center sensor hits obstacle twice, then the big left turn completes, break
    {
      leftSpeed = LEFTSPEED;
      rightSpeed = RIGHTSPEED;

      motors.leftMotor(-leftSpeed);
      motors.rightMotor(-rightSpeed);

      //hit_state from 0 to 1, means hit the obstacle once.
      if( hit_state_now>hit_state_last )
      {
        count0++;
      }
      hit_state_last=hit_state_now;
      if( center.read()<LINETHRESHOLD*0.9 )
        hit_state_now=0;
      if( center.read()>LINETHRESHOLD )
        hit_state_now=1;
    }
  }

  //if hit the obstacle in a proper direction (doesn't need a big left turn)
  //then just turn left slightly
```

```
 leftSpeed = LEFTSPEED;
 rightSpeed = RIGHTSPEED;
 motors.leftMotor(-leftSpeed);
 motors.rightMotor(-rightSpeed);
 while(center.read() > LINETHRESHOLD);

 //After car adjusts the position, compute the goal point in car frame.
 //Phit won't change because of the pure rotation. Here just transform q
 lCount = encoder.getTicks(LEFT);
 rCount = encoder.getTicks(RIGHT);
 lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
 rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
 get_g(g, (lDistance+rDistance)/wheelDistance);
 inv_g(g1, g);
 gq(q, g1, q);

 // Before circumnavigating the obstacle, initialize the S and Dmin
 S=0;//the distance that the car has driven
 Dmin=sqrt(q[1]*q[1]+q[0]*q[0]);//the minimum distance to the goal

 //then start circumnavigating the obstacle
 /* Because the car start from the Phit(hit point), it can't return to the Phit in a short time.
   We set "count" to avoid the car reaching the Phit(hit point) immediately by mistake */
 while( count<300 || (abs(Phit[0])>PHITTHRESHOLD || abs(Phit[1])>PHITTHRESHOLD) )// while not
return to the hit point
 {
  // Tracking
  if(center.read() > LINETHRESHOLD) // if center sensor is above the line, go straight
  {
   leftSpeed = -LEFTSPEED;
   rightSpeed = RIGHTSPEED;
  }
  else if(left.read() > LINETHRESHOLD) // if left sensor is above the line, urn right
  {
   leftSpeed = -(LEFTSPEED - TURNSPEED);
   rightSpeed = RIGHTSPEED + TURNSPEED;
  }
  else if(right.read() > LINETHRESHOLD) // if right sensor is above the line, turn left
  {
   leftSpeed = -(LEFTSPEED + TURNSPEED);
   rightSpeed = RIGHTSPEED - TURNSPEED;
  }

  encoder.clearEnc(BOTH);    //clear the encoder data
  if(left.read()<LINETHRESHOLD || center.read()<LINETHRESHOLD || right.read()<LINETHRESHOLD)
  {
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
  }

  //compute the coordinates of the goal point q
  delay(DELAYTIME);
  lCount = encoder.getTicks(LEFT);
  rCount = encoder.getTicks(RIGHT);
  lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
  rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
  theta = (rDistance - lDistance) / wheelDistance;

  get_g(g, theta, -(lDistance+rDistance)*sin(theta/2)/2, (lDistance+rDistance)*cos(theta/2)/2);
  inv_g(g1, g);
  gq(q, g1, q);
  //then compute the coordinates of Phit(hit point)
  gq(Phit, g1, Phit);

  //update the s first, then see if the Dmin should be updated as well as S1
  S= S+(lDistance+rDistance)/2;
  if( sqrt(q[1]*q[1]+q[0]*q[0])<Dmin )
  {
    Dmin=sqrt(q[1]*q[1]+q[0]*q[0]);
    S1=S;
  }


  /* Because the car start from the Phit(hit point), it can't return to the Phit in a short time.
     We set "count" to avoid the car reaching the Phit(hit point) immediately by mistake */
  if(count<500)
    count++;
}


encoder.clearEnc(BOTH);     //clear the encoder data
//if the distance in current direction to the "P leave" is greater than half of obstacle primeter,
//then move to the gaol in inverse direction for (S-S1)
if(2*S1>S)
{
  //turn by 180°
  while( ((lDistance+rDistance)/wheelDistance)<PI )
  {
    leftSpeed = LEFTSPEED;
    rightSpeed = RIGHTSPEED;
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
    lCount = encoder.getTicks(LEFT);
    rCount = encoder.getTicks(RIGHT);
    lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
    rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
  }
  // circumnavigate the obstacle for (S-S1)
  driveDistance(S-S1);
  //when arrives the "P leave", move towards the goal
  encoder.clearEnc(BOTH);     //clear the encoder data
  lCount = encoder.getTicks(LEFT);
  rCount = encoder.getTicks(RIGHT);
  lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
  rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
  //turn
  while( ((lDistance+rDistance)/wheelDistance)<(PI/2) )
  {
    leftSpeed = LEFTSPEED;
    rightSpeed = RIGHTSPEED;
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
  }

  encoder.clearEnc(BOTH);     //clear the encoder data
  lCount = encoder.getTicks(LEFT);
  rCount = encoder.getTicks(RIGHT);
  lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
  rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
  //move towards
  while( (lDistance+rDistance)<(Dmin*2) )
  {
    leftSpeed = -LEFTSPEED;
    rightSpeed = RIGHTSPEED;
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
  }

}
else  // Otherwise, move to the goal by circumnavigating the obstacle in the current deriction
{
  //circumnavigate the obstacle for S1
```

```
    driveDistance(S1);
    //when arrives the "P leave", move towards the goal
    encoder.clearEnc(BOTH);     //clear the encoder data
    while( ((lDistance+rDistance)/wheelDistance)<(PI/2) )
    {
      leftSpeed = -LEFTSPEED;
      rightSpeed = -RIGHTSPEED;
      motors.leftMotor(-leftSpeed);
      motors.rightMotor(-rightSpeed);
      lCount = encoder.getTicks(LEFT);
      rCount = encoder.getTicks(RIGHT);
      lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
      rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
    }
  }
}

/**************** project1 circumnavigate the obstacle for a certain distance ***************/
void driveDistance(double distance)
{
  int leftSpeed,rightSpeed;
  long stopCount = 0;
  long lCount = 0; // left count value
  long rCount = 0; // right eount value
  double numRev; //goal encoder value
  numRev = distance / wheelCirc; //goal encoude value = goal distance / wheel circumference
  encoder.clearEnc(BOTH);  // clear the encoder count
  while (lCount+rCount < 2*numRev*countsPerRev) // while average count value < goal
  {
    if(center.read() > LINETHRESHOLD) // if center sensor is above the line, go straight
    {
      leftSpeed = -LEFTSPEED;
      rightSpeed = RIGHTSPEED;
    }
        else if(left.read() > LINETHRESHOLD) // if left sensor is above the line, urn right
    {
      leftSpeed = -(LEFTSPEED - TURNSPEED);
      rightSpeed = RIGHTSPEED + TURNSPEED;
    }

    else if(right.read() > LINETHRESHOLD) // if right sensor is above the line, turn left
    {
      leftSpeed = -(LEFTSPEED + TURNSPEED);
      rightSpeed = RIGHTSPEED - TURNSPEED;
    }

    if(left.read()<LINETHRESHOLD || center.read()<LINETHRESHOLD || right.read()<LINETHRESHOLD)
    {
      motors.leftMotor(-leftSpeed);
      motors.rightMotor(-rightSpeed);
    }

    lCount = abs(encoder.getTicks(LEFT)); // get left encoder value
    rCount = abs(encoder.getTicks(RIGHT)); //get right encoder value

  }

}
```

Bug2

```
#include <RedBot.h>
#include <math.h>
#define LED 13
#define BUTTON 12      // button to be pressed to start up
#define LINETHRESHOLD 800
#define LEFTSPEED 60  // sets the nominal speed. Set to any number from 0 - 255.
#define RIGHTSPEED 60
#define TURNSPEED 50
#define countsPerRev 192
#define wheelDiam 65
#define wheelCirc (PI*wheelDiam)
#define delayTime 5
#define DELAYTIME 10
#define wheelDistance 160
#define OFFSET 1
#define PHITTHRESHOLD 150  //permit error that might happen
RedBotSensor left = RedBotSensor(A3);   // initialize a left sensor object on A3
RedBotSensor center = RedBotSensor(A6); // initialize a center sensor object on A6
RedBotSensor right = RedBotSensor(A7);  // initialize a right sensor object on A7
RedBotMotors motors;
RedBotEncoder encoder = RedBotEncoder(A2, 10);
void get_q(double q[2], double x, double y);    // the function to get coordinates of the goal point in car
frame
void get_g(double g[3][3], double angle = 0, double x = 0, double y = 0);// the function to get the g as
```

```
gab
void inv_g(double ginv[3][3], double g[3][3]);// the function to get the g inverse: g1
void gq(double result[2], double g[3][3], double q[2]);// the function to get q' which equals g1*q
void driveLine(double *s, double * q); // the function to circumnavigate the obstacle in bug2

void setup() {
  // put your setup code here, to run once:
  int leftSpeed;   // variable used to store the leftMotor speed
  int rightSpeed;  // variable used to store the rightMotor speed
  long lCount, rCount, lDiff, rDiff, prevlCount, prevrCount;
  double lDistance, rDistance;
  double qA[2] = { 0, 500};
  double s[2] = {0, 0};
  double qB[2], gB1B2[3][3], gB2B1[3][3], gBA[3][3], gAB[3][3];
  double angle;
  Serial.begin(9600);
  pinMode(BUTTON, INPUT_PULLUP);
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);
  get_g(gAB, 0);//angle, x, y
  inv_g(gBA, gAB);
  gq(qB, gBA, qA);
  while(digitalRead(BUTTON) == HIGH);
  Serial.println(qB[0]);
  Serial.println(qB[1]);
  Serial.print("********\n");
  while(abs(qB[0])>50 || abs(qB[1])>50)// while not at the goal
  {
    if (qB[1]>0 && abs(atan(qB[0]/qB[1]))<0.08)// if the goal is almost straight ahead
    {
      // then go straight
      leftSpeed = -LEFTSPEED;
      rightSpeed = RIGHTSPEED;
      motors.leftMotor(-leftSpeed);
      motors.rightMotor(-rightSpeed);
      encoder.clearEnc(BOTH);
      lDistance = 0;
      rDistance = 0;
      prevlCount = 0;
      prevrCount = 0;

      //while not hit obstacle, go strictly straight towards the goal
      while(left.read()<=LINETHRESHOLD        &&        center.read()<=LINETHRESHOLD        &&
right.read()<=LINETHRESHOLD)
```

```
  {
    delay(delayTime);
    lCount = encoder.getTicks(LEFT);
    rCount = encoder.getTicks(RIGHT);
    lCount = abs(lCount);
    rCount = abs(rCount);
    lDistance = double(lCount) * wheelCirc / countsPerRev;
    rDistance = double(rCount) * wheelCirc / countsPerRev;
    if (lDistance + rDistance >= 2*qB[1]) break;//if reaches the goal, break
    lDiff = (lCount - prevlCount);
    rDiff = (rCount - prevrCount);
    prevlCount = lCount;
    prevrCount = rCount;
    if (lDiff > rDiff) //has turned right, should turn left
    {
      leftSpeed += OFFSET;
      rightSpeed += OFFSET;
    }
    // if right is faster than the left, speed up the left / slow down right
    else if (lDiff < rDiff) //has turned left, should turn right
    {
      leftSpeed -= OFFSET;
      rightSpeed -= OFFSET;
    }
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
  }

  qB[1] -= (lDistance + rDistance)/2;//compute the coordinates of the goal q in car frame
  s[1] -= (lDistance + rDistance)/2;//compute the coordinates of the start s in car frame
}
else
{
  if(qB[0] > 0)//if the goal is on the right, then turn right
  {
    //compute the directon of the goal
    if(qB[1] == 0) angle = PI/2;
    else if(qB[1]>0) angle = atan(qB[0]/qB[1]);
    else angle = PI + atan(qB[0]/(qB[1]));
    // turn right until the car face towards the goal point q
    leftSpeed = -LEFTSPEED;
    rightSpeed = -RIGHTSPEED;
    encoder.clearEnc(BOTH);
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
    do
    {
      delay(1);
      lCount = encoder.getTicks(LEFT);
      rCount = encoder.getTicks(RIGHT);
      lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
      rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
    }while (rDistance+lDistance < wheelDistance*angle);
    get_g(gB1B2, -angle);
  }
  else  //otherwise, turn left
  {
    //compute the directon of the goal
    if(qB[1] == 0) angle = PI/2;
    else if(qB[1]>0) angle = atan(-qB[0]/qB[1]);
    else angle = PI + atan(-qB[0]/qB[1]);
    // turn left until the car face towards the goal point q
    leftSpeed = LEFTSPEED;
    rightSpeed = RIGHTSPEED;
    encoder.clearEnc(BOTH);
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
    do
    {
      delay(1);
      lCount = encoder.getTicks(LEFT);
      rCount = encoder.getTicks(RIGHT);
      lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
      rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
    }while (rDistance+lDistance < wheelDistance*angle);
    get_g(gB1B2, angle);
  }

  //compute the start and goal point coordinates expressed in car frame
  qB[1] = sqrt(qB[1]*qB[1]+qB[0]*qB[0]);
  qB[0] = 0;
  inv_g(gB2B1, gB1B2);
  gq(s, gB2B1, s);
}
/********** if hit an obstacle ***********/
if( (center.read()> LINETHRESHOLD) )
{
  //turn left
```

```
      digitalWrite(LED, HIGH);
      driveLine(s, qB);
      digitalWrite(LED, LOW);
    }
  }
  motors.brake();//if reaches the goal, brake.
}


void loop() {
  // put your main code here, to run repeatedly:
}

/*********** the specific function of get_q get_g inv_g gq ************/
void get_q(double q[2], double x, double y)
{
  q[0] = x;
  q[1] = y;
}
void get_g(double g[3][3], double angle, double x, double y)
{
  g[0][0] = cos(angle);
  g[0][1] = -sin(angle);
  g[0][2] = x;
  g[1][0] = sin(angle);
  g[1][1] = cos(angle);
  g[1][2] = y;
  g[2][0] = 0;
  g[2][1] = 0;
  g[2][2] = 1;
}
void inv_g(double ginv[3][3], double g[3][3])
{
  double gg[3][3];
  unsigned char i, j;
  for(i=0; i<3; i++)
  {
    for(j = 0; j<3; j++)
    {
      gg[i][j] = g[i][j];
    }
  }
  ginv[0][0] = gg[0][0];
  ginv[0][1] = gg[1][0];
  ginv[0][2] = -gg[0][0]*gg[0][2] - gg[1][0]*gg[1][2];
  ginv[1][0] = gg[0][1];
  ginv[1][1] = gg[1][1];
  ginv[1][2] = -gg[0][1]*gg[0][2] - gg[1][1]*gg[1][2];
  ginv[2][0] = 0;
  ginv[2][1] = 0;
  ginv[2][2] = 1;
}
void gq(double result[2], double g[3][3], double q[2])
{
  double qq[2] = {q[0], q[1]};
  result[0] = g[0][2] + g[0][0]*qq[0] + g[0][1]*qq[1];
  result[1] = g[1][2] + g[1][0]*qq[0] + g[1][1]*qq[1];
}

/********************** driveLine ***********************/
void driveLine(double*s, double*q)
{
  int hit_state_now=0,hit_state_last=0;
  int count0=0;
  int leftSpeed,rightSpeed;
  long lCount, rCount;
  double lDistance, rDistance;
  double g[3][3], g1[3][3];
  double theta;
  double Phit[2]={ 0,0 };//hit point
  int count=0;
  double angle;
  /************** turn left when hit an abstacle ***************/
  //if the left sensor hits the obstacle first, then the car needs a big left turn
  encoder.clearEnc(BOTH);
  if( left.read()>LINETHRESHOLD&&center.read()<LINETHRESHOLD )//if the left sensor hits the obstacle
first
  {
    while( count0<2 )//if the center sensor hits obstacle twice, then the big left turn completes, break
    {
      leftSpeed = LEFTSPEED;
      rightSpeed = RIGHTSPEED;

      motors.leftMotor(-leftSpeed);
      motors.rightMotor(-rightSpeed);

      //hit_state from 0 to 1, means hit the obstacle once.
      if( hit_state_now>hit_state_last )
      {
        count0++;
      }
      hit_state_last=hit_state_now;
      if( center.read()<LINETHRESHOLD*0.9 )
        hit_state_now=0;
      if( center.read()>LINETHRESHOLD )
        hit_state_now=1;
    }
  }

  //if hit the obstacle in a proper direction (doesn't need a big left turn)
  //then just turn left slightly
  leftSpeed = LEFTSPEED;
  rightSpeed = RIGHTSPEED;
  motors.leftMotor(-leftSpeed);
  motors.rightMotor(-rightSpeed);
  while(center.read() > LINETHRESHOLD);

  //After car adjusts the position, compute the goal point in car frame.
  //Phit won't change because of the pure rotation. Here just transform q
  lCount = encoder.getTicks(LEFT);
  rCount = encoder.getTicks(RIGHT);
  lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
  rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
  get_g(g, (lDistance+rDistance)/wheelDistance);
  inv_g(g1, g);
  gq(q, g1, q);
  gq(s, g1, s);
  //then start circumnavigating the obstacle
  /* Because the car start from the Phit(hit point), it can't return to the Phit in a short time.
     We set "count" to avoid the car reaching the Phit(hit point) immediately by mistake */
  while(s[0]*q[0]>0 || s[1]*q[1]>0 || (s[1]/s[0])/(q[1]/q[0])<0.9 || (s[1]/s[0])/(q[1]/q[0])>1.1)// while not
hit the line
  {
    // Tracking
    if(center.read() > LINETHRESHOLD) // if center sensor is above the line, go straight
    {
      leftSpeed = -LEFTSPEED;
      rightSpeed = RIGHTSPEED;
    }
    else if(left.read() > LINETHRESHOLD) // if left sensor is above the line, urn right
    {
      leftSpeed = -(LEFTSPEED - TURNSPEED);
      rightSpeed = RIGHTSPEED + TURNSPEED;
    }
    else if(right.read() > LINETHRESHOLD) // if right sensor is above the line, turn left
    {
      leftSpeed = -(LEFTSPEED + TURNSPEED);
      rightSpeed = RIGHTSPEED - TURNSPEED;
    }

    encoder.clearEnc(BOTH);     //clear the encoder data
    if(left.read()<LINETHRESHOLD || center.read()<LINETHRESHOLD || right.read()<LINETHRESHOLD)
    {
      motors.leftMotor(-leftSpeed);
      motors.rightMotor(-rightSpeed);
    }

    //compute the coordinates of the goal point q
    delay(DELAYTIME);
    lCount = encoder.getTicks(LEFT);
    rCount = encoder.getTicks(RIGHT);
    lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
    rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
    theta = (rDistance - lDistance) / wheelDistance;

    get_g(g, theta, -(lDistance+rDistance)*sin(theta/2)/2, (lDistance+rDistance)*cos(theta/2)/2);
    inv_g(g1, g);
    gq(q, g1, q);
    gq(s, g1, s);
  }
  encoder.clearEnc(BOTH);     //clear the encoder data
  if(q[0] > 0)//if the goal is on the right, then turn right
  {
    //compute the directon of the goal
    if(q[1] == 0) angle = PI/2;
    else if(q[1]>0) angle = atan(q[0]/q[1]);
    else angle = PI + atan(q[0]/(q[1]));
    // turn right until the car face towards the goal point q
    leftSpeed = -LEFTSPEED;
    rightSpeed = -RIGHTSPEED;
    encoder.clearEnc(BOTH);
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
    do
    {
      delay(1);
```

```
      lCount = encoder.getTicks(LEFT);
      rCount = encoder.getTicks(RIGHT);
      lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
      rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
    }while (rDistance+lDistance < wheelDistance*angle);
    get_g(g, -angle);
  }
  else  //otherwise, turn left
  {
    //compute the directon of the goal
    if(q[1] == 0) angle = PI/2;
    else if(q[1]>0) angle = atan(-q[0]/q[1]);
    else angle = PI + atan(-q[0]/q[1]);
    // turn left until the car face towards the goal point q
    leftSpeed = LEFTSPEED;
    rightSpeed = RIGHTSPEED;
    encoder.clearEnc(BOTH);
    motors.leftMotor(-leftSpeed);
    motors.rightMotor(-rightSpeed);
    do
    {
      delay(1);
      lCount = encoder.getTicks(LEFT);
      rCount = encoder.getTicks(RIGHT);
      lDistance = abs(double(lCount)) * wheelCirc / countsPerRev;
      rDistance = abs(double(rCount)) * wheelCirc / countsPerRev;
     }while (rDistance+lDistance < wheelDistance*angle);
     get_g(g, angle);
  }

      //compute the start and goal point coordinates expressed in car frame
      q[1] = sqrt(q[1]*q[1]+q[0]*q[0]);
      q[0] = 0;
      inv_g(g1, g);
      gq(s, g1, s);
}
```