Autonomous Robot: Path planning

Guillaume Lemaître - Sophia Bano
Heriot-Watt University, Universitat de Girona, Université de Bourgogne
g.lemaitre58@gmail.com - sophiabano@hotmail.com

I. Objective

The main objective of this laboratory exercise was to program the e-puck Robot for path planning behaviour using Webots Environment . The path planning behaviour has to be implemented using bug 2 algorithm.

II. Introduction

In order to solve path planning problems, Bug algorithms are used which are the simplest sensor-based planner. The Bug algorithm family are well-known robot navigation algorithms with proven termination conditions for unknown environments [1] [2].

The general idea of bug 2 algorithm is that move the robot towards the goal, unless an obstacle is encountered,, in which case, circumnavigate the robot until motion toward the goal is once again allowable. Thus in Bug 2 algorithm a line is first drawn, which is called m-line, from start point to goal point. The robot tries to follow this line from start of program. Thus robot align itself over this line. Now if an obstacle is detected, robot leaves this line and start following the obstacle, until it detects m-line again. At this stage, robot starts following m-line again as shown in Figure 1

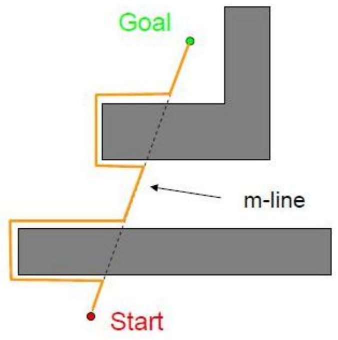


Figure 1. General Diagram illustrating Bug 2 Algorithm

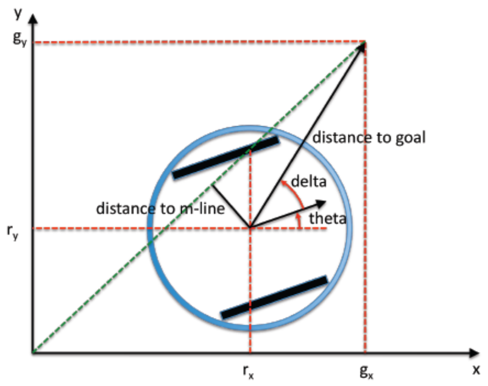


Figure 2. Geometric representation of the robot

III. Theory and implementation

A. Geometry

In order to plan the path which will allow to achieve the goal, the three different measures will be computed.

- Distance to the goal: Euclidean distance between the centre of the robot and the goal position.
- Delta: angle between the x axis of the robot and the goal position.
- Distance to the m-line: Euclidean distance between the centre of the robot and the m-line.

The figure 2 shows the geometric representation of these different distances and angles.

The following parts will introduce the mathematical tools permitting to compute these different parameters. All variables used are presented on the figure 2.

1) Distance from the robot to the goal: The formula which allows to compute the Euclidean distance between the center of the robot and the goal position is:

$$D_{goal} = \sqrt{(g_x + r_x)^2 + (g_y + r_y)^2} \qquad (1)$$

The code implementing this formula is presented below:

```
static double eucl_dist_rob_goal(double gx
    , double gy, double rx, double ry)
{
        //Compute the Euclidean distance
            between the robot and the goal
             point
        return sqrt((gx-rx)*(gx-rx)+(gy-ry
            )*(gy-ry));
}
```

2) Angle from the robot to the goal: The formula which allows to compute the angle between the center of the robot and the goal positiond is:

$$\delta = \alpha - \theta \tag{2}$$

where

$$\alpha = \arctan \frac{g_y - r_y}{g_x - r_x} \tag{3}$$

and θ is given by the odometry as the angle between the horizontal and the x axis of the robot

The code implementing this formula is presented below:

```
static double delta_xrob_goal(double gx,
    double gy, double rx, double ry,
    double theta)
{
        double alpha = 0;
        alpha = ((atan2((gy-ry),(gx-rx)))
            *180)/3.14159;
  printf("\n alpha = %.1lf",alpha);
  //Wrap theta
  theta_wrap=theta*180/3.141592;
  //printf("\n theta = %.1lf",theta);
  if(theta_wrap >180)
  {
    theta_wrap=theta_wrap-360;
  }
  else if (theta_wrap < -180)
  {
    theta_wrap=theta_wrap+360;
  }

  double vac = alpha - theta_wrap;


  printf("\n theta_wrap = %.1lf",
      theta_wrap);
  printf("\n vac = %.1lf",vac);

        return vac;
}
```

3) Distance from the robot to the m-line: The formula which allows to compute the Euclidean distance between the center of the robot and the m-line:

$$D_{m-line} = \left| \frac{r_x \times g_y - r_y \times g_x}{\sqrt{g_y^2 + g_x^2}} \right| \tag{4}$$

and θ is given by the odometry as the angle between the horizontal and the x axis of the robot

The code implementing this formula is presented below:

```
static double eucl_dist_rob_mline(double
    rx, double ry, double gx, double gy)
{
        //Compute the Euclidean distance
  double vac = rx*gy-ry*gx;
  if (vac > 0)
        return ((rx*gy-ry*gx)/(sqrt(gy*gy+
            gx*gx)));
  else
  return (-(rx*gy-ry*gx)/(sqrt(gy*gy+gx*gx
      )));
}
```

B. Implementing Path Planning - Bug 2

There are two modes of operations or behaviours of Robot which has to be implemented in order to develop Bug 2 algorithm using e-puck Robot.

- Head Towards Goal Behaviour
- Obstacle Following Behaviour

1) Head Towards Goal Behaviour: The aim of this step of the algorithm is to approach the m-line and follows it until the goal. However, if an obstacle is found on the trajectory, the robot will follow the object until to find again the m-line. This last part is implemented in the next section.

Four different behaviours are implemented:

- The robot is not oriented to the direction of the goal

When the robot is not oriented to the direction of the goal, when δ is greater than 20°, the robot will only rotate with the maximum angle speed (60.0).

- The robot is oriented to the direction of the goal

When the robot is oriented to the direction of the goal, when δ is inferior than 20°, the robot will go in the direction of the goal with a linear speed important

(100.0) and an angle speed proportional to δ as the angular speed is equal at three times δ.

- The robot is near of the goal

When the robot arrives near of the object and that the Euclidean distance computes is less than 2 centimeters, the robot reach is final position and the angular and linear speed are equal to zero.

- The robot meet an obstacle

When the robot meet an obstacle to the front of it, the object following behaviour is validate and the state 1 is activated.

The following code allows to implement the previous behaviour described.

```
// //////////////////////
// Initialisation state
// Go in ahead until to
// find an object
// When an object is
// detected go to the
// state 1
// //////////////////////
if(state==0)
{
  //// If ab object is detected
  if((sensors_value[0]<thres)||(
      sensors_value[1]<thres) ||(
      sensors_value[6]<thres) ||(
      sensors_value[7] < thres))
  {
    //// Go to the next state
    printf("Detected␣object␣−>␣state␣1␣\n"
        );
    state=1;
  }
  //// Otherwise
  else
  {
    //// Go ahead
    printf("Going␣ahead␣\n");
    state=0;

    if (delta>thres_angle)
    {
      linear_speed=0.0;
      angular_speed=60.0;
    }
    else if  (delta< (−thres_angle))
    {
      linear_speed=0.0;
      angular_speed=−60.0;
    }
    else
    {
      linear_speed=100.0;
      angular_speed=(3∗delta);
    }

    if (eudist<0.02)
    {
      linear_speed=0;
      angular_speed=0;
      printf("\n␣mline␣achieved␣=␣%␣.2lf",
          dist_mline);
    }

    flag = 1;
  }
}
```

2) *Obstacle Following Behaviour:* Once the Head towards goal behaviour is implemented, next task is to implement obstacle following behaviour which takes into account head towards goal behaviour as well. Obstacle following behaviour has already been implemented in laboratory exercise 4. To merge these two behaviours following modifications has been done in obstacle following code:

- **State 0**

If the robot does not sense anything in front of it, it will keep on following the mline as discussed in previous section. But once it detects obstacle it will jump to state 1. This is shown in Figure 3(a).

- **State 1**

Once the robot sense obstacle in front, it will jump to this state. The robot will keep on turning left until sensor 2 detects obstacle and sensor 0 and 1 does not detect any obstacle. At this point robot will jump to state 2 but there are two conditions for path planning which has to be considered.

- Robot is not on mline and it jumps to state 2.
- Robot is on mline and it jumps to state 2.

To distinguish between these two condition flag2 is used. If the robot is already on mline flag2 is set to 1, else flag2 will be zero. The general algorithm is summarized in Figure 3(b). The modified code for this state is as following:

```
// ///////////////////////////
// State to put
//the robot perpendicular
// to an obstacle
// and in the same time
// that no obstacle are
// on the front of the robot
// //////////////////////////
if(state == 1)
{
    ///// If the robot is perpendicular to
        one obstacle
    ///// And that the front of the robot is
        free
    if((sensors_value[2] <(thres+0.3)) && ((
        sensors_value[1]>thres-0.3)&&(
        sensors_value[0]>thres)))
    {
      //// Go to the state 2
      printf("\n␣Perpendicular␣to␣the␣object
          ␣and␣free␣field␣at␣the␣front␣->␣
          state␣2␣\n");
      state=2;
      if ((dist_mline*100) < mline_thres)
      {
        flag2=1;
      }
    }
    ///// Otherwise
    else
    {
      //// Turn left
      printf("\n␣Turning␣left␣\n");
      state=1;
      linear_speed=0.0;
      angular_speed=100.0;
    }
}
```

- **State 2**

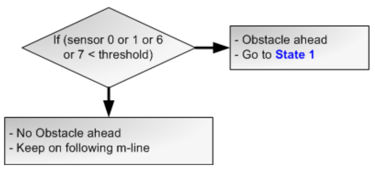
When in State 2, robot will keep on following the obstacle until it detects an mline. If flag2 = 1 at the beginning of state 2, this means that robot is already on mline when it found an obstacle and now it has to follow obstacle. So it follows obstacle and starts moving away from mline. When if moves a distance of mline threshold away from mline , status of flag2 is changes to zero again. This shows that robot is no longer on mline.

The robot will follow the obstacle until it detects mline again. At this point robot will take a hard left turn so that sensor 2 no longer remain close to obstacle. At this stage robot will jump to state 0 again. The general algorithm is summarized in Figure 3(c). The modified code for this state is as following:

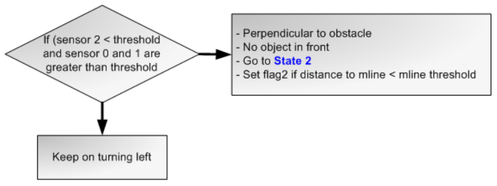
```
///////////////////////////////////
// Following the
// obstacle the nearest
// possible
// Until the moment that
// the robot detect
// something at
// the front -> Return to
// the state 1 to be perpendicular
// and have a free front range
///////////////////////////////////
if(state==2)
{
  //// If the robot is perpendicular and
      near of the obstacle
  if(sensors_value[2]<thres+0.5)
  {
    ////////////////////////////
    ///// Definition of variables
    ///// Definition of the slope
    double m=10;
    /////  Parameter for the linear
        function to use the angle
    double p1 = 5;
    double p2 = 25;
    double p3 = 10;
    ///////////////////////////
    ///// Compute the distance between the
        sensor at 90
    ///// and the sensors at 45
    double d=sensors_value[1]-
        sensors_value[2];
    /////////////////////////
    /////////////////////////
    ///// Function to find the minimum
        distance between
    ///// the sensor at 90  and 45
    double min_sensor = 0;
    if (sensors_value[1]>sensors_value[2])
    {
      min_sensor = sensors_value[2];
    }
    else
    {
      min_sensor = sensors_value[1];
    }
    /////////////////////////

    ////////////////////////
    // If the distance between
    //both sensor is so important
    // It seems that the robot
    // is an corner of the obtacle
    // and has to turn right
    ////////////////////////
    if(d>1)
    {
      /////////////////////
      // We have to compute
      // the angular speed
      // depending
      // of the distance of
      // where the robot is from
      // the border of the obstacle
      ////////////////////

      ////////////////////
      // if the robot is near
      // of the corner
      // turn the softest as
      // possible
      ////////////////////
    if (min_sensor<thres - 0.5)
    {
      linear_speed=80.0;
      angular_speed=-((m*min_sensor)+p1)
          ;
      printf(”\n␣Corner␣near:␣Turn␣soft␣
          right␣\n”);
    }
    //////////////////////
    // if the robot is going
    // to far of the corner
    // Turn more to the right
    //////////////////////
    else if (min_sensor > thres)
    {
      linear_speed=80.0;
      angular_speed=-((m*min_sensor)+p2)
          ;
      printf(”\n␣Corner␣far:␣Turn␣hard␣
          right␣\n”);
    }
    //////////////////
    //// Otherwise
    //////////////////
    else
    {
      linear_speed=80.0;
      angular_speed=-((m*d)+p3);
      printf(”\n␣Turn␣right␣\n”);
    }
    flag = 0;
  }
  ///////////////////
  // If the robot follow
  // the edge but is to near of
  // the edge -> turn left
  ///////////////////
  else if((d<=1) && (sensors_value[2]<
      thres))
  {
    printf(”\n␣Too␣near␣of␣the␣edge␣->␣
        turn␣left␣\n”);
    linear_speed=80.0;
    angular_speed=20.0;
    flag = 0;
  }
  ///////////////////
  // Otherwise
  // Go ahead
  ///////////////////
  else
  {
    printf(”\n␣Going␣ahead␣\n”);
    linear_speed=80.0;
    angular_speed=0.0;
    flag = 0;
  }


  ///////////////////
  //// If we detect
    //an object at the
    // front of the robot
    // -> state 1 to be
    // perpendicular and have
    // free front range
    ///////////////////
    if((sensors_value[0]<thres)||(
        sensors_value[7]<thres))
    {
      printf(”\n␣Object␣at␣the␣front␣->␣
          state␣1␣\n”);
      state = 1;
    }
  }

  ///////////////////
  // If the sensors
  //of the side have
  //big values, it means
  // that the robot doesn't
  //follow anymore an obstacle
  // -> Go to the state 0
  // to find a new obstacle
  ///////////////////
  else if((sensors_value[1]>=3.5) && (
      sensors_value[2]>=3.5))
  {
    linear_speed=80.0;
    angular_speed=0.0;
    state=0;
    printf(”\n␣Loose␣the␣object␣->␣state␣0
        ␣\n”);
  }

  if (((dist_mline*100) < mline_thres)&&(
      flag == 0)&&(flag2==0))
  {
    printf(”\n␣Detect␣the␣m-line␣->␣turn␣
        left␣hard␣\n”);
    linear_speed=0.0;
    angular_speed=200.0;
    state = 0;
    flag = 1;
  }
  if(((dist_mline*100) >= mline_thres)&&(
      flag2==1))
  {
    flag2=0;
  }

}
```

IV. RESULTS

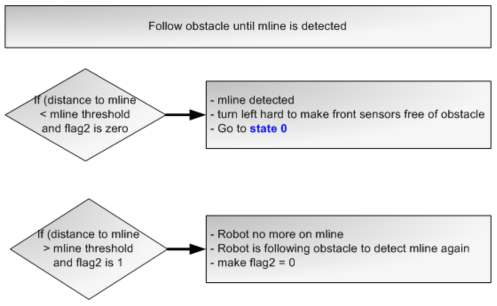
Results for three different mline and obstacles in way to mline are shown in Figure 4, 5 and 6. The mline is shown in red, while the path followed by robot is shown in blue colour. Note that in all three results shown robot successfully follow the two behaviours of Bug 2


If (sensor 0 or 1 or 6 or 7 < threshold)
- Obstacle ahead
- Go to State 1
- No Obstacle ahead
- Keep on following m-line

(a) State 0


If (sensor 2 < threshold and sensor 0 and 1 are greater than threshold
- Perpendicular to obstacle
- No object in front
- Go to State 2
- Set flag2 if distance to mline < mline threshold
Keep on turning left

(b) State 1


Follow obstacle until mline is detected
If (distance to mline < mline threshold and flag2 is zero
- mline detected
- turn left hard to make front sensors free of obstacle
- Go to state 0
If (distance to mline > mline threshold and flag2 is 1
- Robot no more on mline
- Robot is following obstacle to detect mline again
- make flag2 = 0

(c) State 2

Figure 3. Flow of each state after implementing Path Planning Algorithm

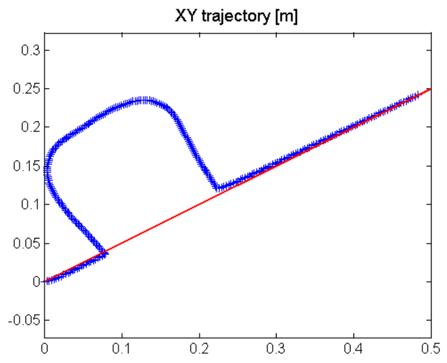


Figure 4. Path Planning Result with a single obstacle in way of mline

algorithm, i-e, it successfully go ahead towards the goal following mline and if it finds an obstacle in its way,it circumnavigate around the obstacle until it finds mline again.

V. Conclusion

Bug 2 path planning algorithm has been successfully implemented. Results shows that our developed algorithm is capable of achieving both behaviours of Bug 2 algorithm accurately.

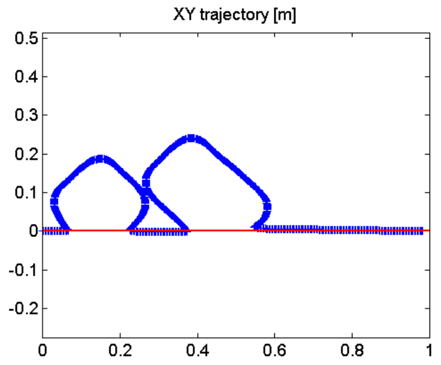


Figure 5. Path Planning Result with two obstacles in way of mline

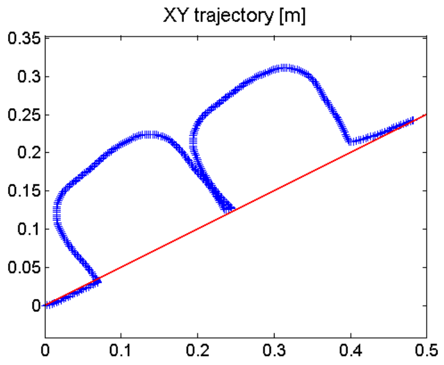


Figure 6. Path Planning Result with two difficult obstacle within Robot way

References

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, June 2005.

[2] V. Lumelsky and A. Stepanov, “Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape,” *Algorithmica*, vol. 2, no. 1, pp. 403–430, 1987.

Appendix A
Code

```c
#include <webots/robot.h>
#include <webots/differential_wheels.h>
#include <webots/distance_sensor.h>
#include <webots/light_sensor.h>
#include <webots/camera.h>
#include <webots/accelerometer.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
#include <stdio.h>

#define TIME_STEP 256
#define WHEEL_RADIUS 0.0205
#define AXLE_LENGTH 0.053


/* GLOBAL VARIABLES */

double sim_time,real_time;
double l_past,r_past;
double dl,dr,d,da;
double theta,x,y;
double sensors_value[8];
double linear_speed,angular_speed;
double left_speed, right_speed;
double displacement;
double IR2dist=0;
int count=0;
int flag=0;
int flag2=0;
double theta_wrap;

double state, iniX, iniY, iniTheta;

double gx=2;
double gy=0;
double eudist,delta,dist_mline;
double thres_angle=20;
float mline_thres=2;


static void compute_odometry() {
  double l = wb_differential_wheels_get_left_encoder();
  double r = wb_differential_wheels_get_right_encoder();

  dl = ((l-l_past) / 1000.0) * 2 * 3.141592* WHEEL_RADIUS; // distance covered by left wheel
       in meter
  dr = ((r-r_past) / 1000.0) * 2 * 3.141592* WHEEL_RADIUS;  // distance covered by right
      wheel in meter
  d  = (dr + dl) / 2 ;
  da = (dr - dl) / AXLE_LENGTH; // delta orientation
  r_past=r;
  l_past=l;
  displacement=displacement+(d*100);
  //printf("estimated distance covered by left wheel: %g m.\n",dl);
  //printf("estimated distance covered by right wheel: %g m.\n",dr);
  printf("estimated distance covered by the robot: %g m.\n",d);
  //printf("estimated change of orientation: %g rad.\n",da);
  //printf("Wheel distances: l= %f; r= %f; dl= %f; dr= %f; d= %f; da= %f;\n",l,r,dl,dr,d,da)
      ;


  theta=theta+da;
  x=x+d*cos(theta);
  y=y+d*sin(theta);

  printf("displacement: %f \n",displacement);

}



static double eucl_dist_rob_goal(double gx, double gy, double rx, double ry)
{
        //Compute the euclidean distance between the robot and the goal point
        return sqrt((gx-rx)*(gx-rx)+(gy-ry)*(gy-ry));
}

static double delta_xrob_goal(double gx, double gy, double rx, double ry, double theta)
{
        double alpha = 0;
        alpha = ((atan2((gy-ry),(gx-rx)))*180)/3.14159;
  printf("\n alpha = %.1lf",alpha);
  //Wrap theta
  theta_wrap=theta*180/3.141592;
  //printf("\n theta = %.1lf",theta);
  if(theta_wrap >180)
  {
    theta_wrap=theta_wrap-360;
  }
  else if (theta_wrap < -180)
  {
    theta_wrap=theta_wrap+360;
  }

  double vac = alpha - theta_wrap;


  printf("\n theta wrap = %.1lf",theta_wrap);
  printf("\n vac = %.1lf",vac);

        return vac;
}

static double eucl_dist_rob_mline(double rx, double ry, double gx, double gy)
{
        //Compute the euclidean distance
  double vac = rx*gy-ry*gx;
  if (vac > 0)
        return ((rx*gy-ry*gx)/(sqrt(gy*gy+gx*gx)));
  else
  return (-(rx*gy-ry*gx)/(sqrt(gy*gy+gx*gx)));
}

static void high_level_controller() {


///////////////////////////////////////////////
//// Definition of parameters
```

```
///////////////////////////////////////////////
double thres=1.5;

eudist=eucl_dist_rob_goal(gx,gy,x,y);
delta=delta_xrob_goal(gx,gy,x,y,theta);
dist_mline=eucl_dist_rob_mline(gx,gy,x,y);

printf("\n flag : %d \n",flag);


/////////////////////////////////////////////////////////
///// Initialisation state
///// Go in ahead until to find an object
///// When an object is detected go to the state 1
/////////////////////////////////////////////////////////
if(state==0)
{
  //// If ab object is detected
  if((sensors_value[0]<thres)||(sensors_value[1]<thres) ||(sensors_value[6]<thres) ||(
      sensors_value[7] < thres))
  {
    //// Go to the next state
    printf("Detected object -> state 1 \n");
    state=1;
  }
  //// Otherwise
  else
  {
    //// Go ahead
    printf("Going ahead \n");
    state=0;

    if (delta>thres_angle)
    {
      linear_speed=0.0;
      angular_speed=60.0;
    }
    else if  (delta< (-thres_angle))
    {
      linear_speed=0.0;
      angular_speed=-60.0;
    }
    else
    {
      linear_speed=100.0;
      angular_speed=(3*delta);
    }

    if (eudist<0.02)
    {
      linear_speed=0;
      angular_speed=0;
      printf("\n mline achieved = % .2lf",dist_mline);
    }
    //printf("\ndelta = %.2lf\n",delta);
    //printf("dist goal = %.2lf\n",eudist);
    //printf("dist mline = %.2lf\n",dist_mline);

    flag = 1;

    //linear_speed=80.0;
```

```
      //angular_speed=0.0;
    }
}

//////////////////////////////////////////////////////////////////
///// State to put the robot perpendicular to an obstacle
///// and in the same time that no obstacle are on the
///// front of the robot
//////////////////////////////////////////////////////////////////
if(state == 1)
{
  ///// If the robot is perpendicular to one obstacle
  ///// And that the front of the robot is free
  if((sensors_value[2] <(thres+0.3)) && ((sensors_value[1]>thres-0.3)&&(sensors_value[0]>
      thres)))
  {
    //// Go to the state 2
    printf("\n␣Perpendicular␣to␣the␣object␣and␣free␣field␣at␣the␣front␣->␣state␣2␣\n");
    state=2;
    if ((dist_mline*100) < mline_thres)
    {
      flag2=1;
    }
  }
  ///// Otherwise
  else
  {
    //// Turn left
    printf("\n␣Turning␣left␣\n");
    state=1;
    linear_speed=0.0;
    angular_speed=100.0;
  }
}

////////////////////////////////////////////////////////////////////
///// Following the obstacle the nearest possible
///// Until the moment that the robot detect something at
///// the front -> Return to the state 1 to be perpendicular
///// and have a free front range
////////////////////////////////////////////////////////////////////

if(state==2)
{
  //// If the robot is perpendicular and near of the obstacle
  if(sensors_value[2]<thres+0.5)
  {
    //////////////////////////////////////////////////////////////
    ///// Definition of variables
    ///// Definition of the slope
    double m=10;
    /////  Parameter for the linear function to use the angle
    double p1 = 5;
    double p2 = 25;
    double p3 = 10;
    //////////////////////////////////////////////////////////////
    ///// Compute the distance between the sensor at 90
    ///// and the sensors at 45
    double d=sensors_value[1]-sensors_value[2];
    //////////////////////////////////////////////////////////////
    //////////////////////////////////////////////////////////////
```

```
///// Function to find the minimum distance between
///// the sensor at 90  and 45
double min_sensor = 0;
if (sensors_value[1]>sensors_value[2])
{
  min_sensor = sensors_value[2];
}
else
{
  min_sensor = sensors_value[1];
}
///////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////
///// If the distance between both sensor is so important
///// It seems that the robot is an corner of the obtacle
///// and has to turn right
///////////////////////////////////////////////////////////
if(d>1)
{
  /////////////////////////////////////////////////////////
  //// We have to compute the angular speed depending
  //// of the distance of where the robot is from
  //// the border of the obstacle
  /////////////////////////////////////////////////////////

  /////////////////////////////////////////////////////////
  //// if the robot is near of the corner
  //// turn the softest as possible
  ////////////////////////////////////////////////////////
  if (min_sensor<thres - 0.5)
  {
    linear_speed=80.0;
    angular_speed=-((m*min_sensor)+p1);
    printf("\n␣Corner␣near:␣Turn␣soft␣right␣\n");
  }
  /////////////////////////////////////////////////////////
  //// if the robot is going to far of the corner
  //// Turn more to the right
  /////////////////////////////////////////////////////////
  else if (min_sensor > thres)
  {
    linear_speed=80.0;
    angular_speed=-((m*min_sensor)+p2);
    printf("\n␣Corner␣far:␣Turn␣hard␣right␣\n");
  }
  ///////////////////////////////////////////////////////////
  //// Otherwise
  ///////////////////////////////////////////////////////////
  else
  {
    linear_speed=80.0;
    angular_speed=-((m*d)+p3);
    printf("\n␣Turn␣right␣\n");
  }
  flag = 0;
}
///////////////////////////////////////////////////////////
//// If the robot follow the edge but is to near of
//// the edge -> turn left
///////////////////////////////////////////////////////////
```

```
    else if((d<=1) && (sensors_value[2]<thres))
    {
      printf("\n Too near of the edge -> turn left \n");
      linear_speed=80.0;
      angular_speed=20.0;
      flag = 0;
    }
    /////////////////////////////////////////////////////////////
    ///// Otherwise
    ///// Go ahead
    /////////////////////////////////////////////////////////////
    else
    {
      printf("\n Going ahead \n");
      linear_speed=80.0;
      angular_speed=0.0;
      flag = 0;
    }


    ////////////////////////////////////////////////////////////
    //// If we detect an object at the front of the robot
    //// -> state 1 to be perpendicular and have
    //// free front range
    ////////////////////////////////////////////////////////////
    if((sensors_value[0]<thres)||(sensors_value[7]<thres))
    {
      printf("\n Object at the front -> state 1 \n");
      state = 1;
    }
  }

  ////////////////////////////////////////////////////////////////
  //// If the sensors of the side have big values, it means
  //// that the robot doesn't follow anymore an obstacle
  //// -> Go to the state 0 to find a new obstacle
  ////////////////////////////////////////////////////////////////
  else if((sensors_value[1]>=3.5) && (sensors_value[2]>=3.5))
  {
    linear_speed=80.0;
    angular_speed=0.0;
    state=0;
    printf("\n Loose the object -> state 0 \n");
  }

  if (((dist_mline*100) < mline_thres)&&(flag == 0)&&(flag2==0))
  {
    printf("\n Detect the m-line -> turn left hard \n");
    linear_speed=0.0;
    angular_speed=200.0;
    state = 0;
    flag = 1;
  }
  if(((dist_mline*100) >= mline_thres)&&(flag2==1))
  {
    flag2=0;
  }

}
```

```
printf("\n State=%.1lf\n",state);


}


 static  void low_level_controller() {

   left_speed=linear_speed-angular_speed;
   right_speed=linear_speed+angular_speed;

}

 static  void update_log_file() {

   int i;
   FILE * logFile;

   logFile = fopen("logFile.txt","a");
   fprintf(logFile, "%f %f %f %f %f",real_time,sim_time,x,y,theta);

   for (i = 0; i < 8; i++) {
       fprintf(logFile, " %f",sensors_value[i]);
     }
   fprintf(logFile, " %f %f %f %f \n",linear_speed,angular_speed,left_speed,right_speed);

   fclose ( logFile );
   printf("Time: real=%0.3fs; sim=%0.3fs. \nPosition: x= %0.3fm; y= %0.3fm; theta= %0.3
       fdegree; \n",real_time,sim_time,x,y,theta*180/3.141592);
   printf("Distance sensors: %0.1f / %0.1f / %0.1f / %0.1f / %0.1f / %0.1f / %0.1f / %0.1f\n"
       ,sensors_value[0],sensors_value[1],sensors_value[2],sensors_value[3],sensors_value[4],
       sensors_value[5],sensors_value[6],sensors_value[7]);
   printf("Linear speed: %0.1f; Angular speed: %0.1f;\n\n",linear_speed,angular_speed);


}

 int main(int argc, char *argv[]) {

   /* define variables */
   WbDeviceTag distance_sensor[8];
   double coef[8]={0};
   double value=0;
   int i;

   clock_t start, end;

   /* initialize Webots */
   wb_robot_init();

   /* get and enable devices */
   wb_differential_wheels_enable_encoders(TIME_STEP);


   for (i = 0; i < 8; i++) {
     char device_name[4];

     /* get distance sensors */
     sprintf(device_name, "ps%d", i);
     distance_sensor[i] = wb_robot_get_device(device_name);
     wb_distance_sensor_enable(distance_sensor[i],TIME_STEP);
```

```
  }

  /*initialize global variables*/
  sim_time=0.0;
  real_time=0.0;
  end = clock();
  theta=0.0;
  x=0.0;
  y=0.0;
  wb_robot_step(TIME_STEP);
  l_past = wb_differential_wheels_get_left_encoder();
  r_past = wb_differential_wheels_get_right_encoder();

  state=0;
  iniX = x;
  iniY = y;
  iniTheta = theta;
  count=0;
  displacement=0;
  state=0;
  /* main loop */
  for (;;) {
coef[8]=1.8658*pow(10,-27);
coef[7]=-3.0795*pow(10,-23);
coef[6]=2.1084*pow(10,-19);
coef[5]=-7.7546*pow(10,-16);
coef[4]=1.6596*pow(10,-12);
coef[3]=-2.0957*pow(10,-9);
coef[2]=1.5174*pow(10,-6);
coef[1]=-0.00058968*pow(10,0);
coef[0]=0.1154*pow(10,0);  // real p0=[-2.8796e-24,4.0641e-20,-2.309e-16,6.7637e-13,-1.0882e
    -09,9.5095e-07,-0.0004,0.0951]
  // sim p0 = [-3.5638e-24, 4.3924e-20, -2.1927e-16, 5.6962e-13, -8.2424e-10,6.6189e-07,
      -0.0003,0.06797]
  for (i = 0; i < 8; i++) {

  value = wb_distance_sensor_get_value(distance_sensor[i]);

  if (value >4000)
  {
  value=4000;
  }
  else if(value<130)
  {
  value=130;
  }
  sensors_value[i] = coef[8]*pow(value,8)+coef[7]*pow(value,7)+coef[6]*pow(value,6)+coef[5]*
      pow(value,5)+coef[4]*pow(value,4)+coef[3]*pow(value,3)+coef[2]*pow(value,2)+coef[1]*
      pow(value,1)+coef[0];
  sensors_value[i]=sensors_value[i]*100; //m to cm
  if (sensors_value[i]<0)
  {
   sensors_value[i]=0;
  }
  printf("sensors_values=␣%lf␣\n",sensors_value[i]);
  }


      /* get sensors values */
   // for (i = 0; i < 8; i++) {
   //   sensors_value[i] = wb_distance_sensor_get_value(distance_sensor[i]);
```

```
      //printf("distance %f \n",sensors_value[i]);
    // }

     /* compute odometry*/
     compute_odometry();

     /* compute high-level control */
     high_level_controller();
/*
     eudist=eucl_dist_rob_goal(gx,gy,x,y);
     printf("\n theta = %.1lf",theta);
     delta=delta_xrob_goal(gx,gy,x,y,theta);
     dist_mline=eucl_dist_rob_mline(gx,gy,x,y);

     if (delta>thres_angle)
     {
       linear_speed=0.0;
       angular_speed=60.0;
     }
     else if  (delta< (-thres_angle))
     {
       linear_speed=0.0;
       angular_speed=-60.0;
     }
     else
     {
       linear_speed=100.0;
       angular_speed=(3*delta);
     }

     if (eudist <0.02)
     {
       linear_speed=0;
       angular_speed=0;
       printf("\n mline achieved = % .2lf",dist_mline);
     }

      */
     printf("\ndelta␣=␣%.2lf\n",delta);
     printf("dist␣goal␣=␣%.2lf\n",eudist);
     printf("dist␣mline␣=␣%.2lf\n",dist_mline*100);


     /* compute low-level control */
     low_level_controller();

     /* set speed values */
     wb_differential_wheels_set_speed(left_speed,right_speed);

     /* save data in a log file */
     update_log_file();

     /* perform a simulation step */
     wb_robot_step(TIME_STEP);


     /* updating the real and simulating time */
     double cpu_time_used;
     start = clock();
     cpu_time_used = ((double) (start-end)) / CLOCKS_PER_SEC;
     end = start;
    real_time = real_time + cpu_time_used;
    sim_time = sim_time + TIME_STEP/1000.0;
  }

  return 0;
}
```