E-Puck

Submitted by

Antony AKINBOH
JAMPY Florian
Sai Krishna PATHI

Objective:

The goal of this lab exercise is to learn how to program e-puck, which is based on webot software. The main objectives of this exercise are

- To stop in front of the wall or obstacle
- To make a square
- Collision Avoidance
- Obstacle Follower
- Bug Algorithm (To reach the goal)

Procedure:

First we set the e-puck's default speed by the given statements

```
//default instruction
speed[0] = 200.0;
speed[1] = 200.0;
```

For all the operations to be achieved, we considered IR sensors. The Figure 1 shows the IR sensors location on e-puck is shown below:

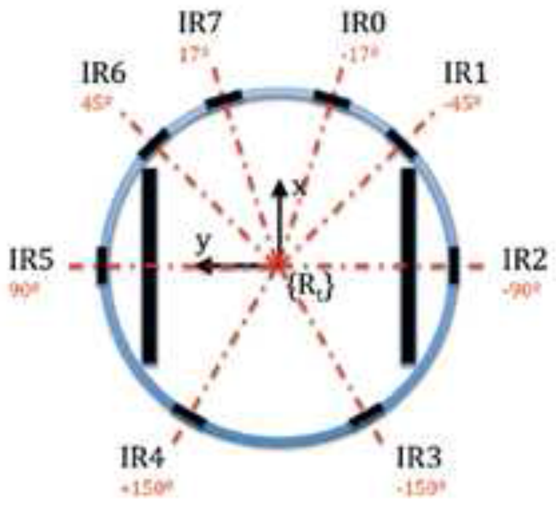

Figure 1

To Stop in front of the wall or obstacle:

```
if(sensors_value[0]>500||sensors_value[7]>500)
{
    speed[0]=0.0;
    speed[1]=0.0;
}
```

We did this task by using “ if ” condition to check the proximity sensor response against distance. If the distance is 2 cms then we stop the robot by setting the speed of the left and right wheel to Zero by using the sensor response obtained.

To make a square:

We did this task by considering the encoders, the orientation and the distance moved by them.

```
if(dang>=1.57)
  {
    wb_differential_wheels_set_encoders(0,0);
  }
if(rightw>=1900)
  {
   speed[0]=-200;
   speed[1]=200;
  }
if(dang>=1.57)
  {
   wb_differential_wheels_set_encoders(0,0);
  }
```

In the above, if the wheels cover a particular distance then we set them to turn to left. Again we move and we do the same process to make a square.

Collision Avoidance:

```
if
(sensors_value[0]>500||sensors_value[7]>500||sensors_value[6]>
500||sensors_value[1]>500)
    {
      speed[0]=150.0;
      speed[1]=-150.0;
      Wall=1;
    }
```

Here, we took sensors 0, 1, 6, 7 into consideration and checked their sensor response. And if we are too close from the wall then the speed of the left wheel is set to positive and the speed of the right is set to negative as shown above code. With this instruction we avoid the wall and keep it to the left side. Wall is a Boolean to know if we have encountered an obstacle (a wall) at least one time.

Obstacle Follower:

To do this task, first we use the collision avoidance condition and then we use the following " if " condition

```
 if (sensors_value[5]<=400&&sensors_value[5]>=350)
   {
      speed[0]=100.0;
      speed[1]=150.0;
      Wall=1;
    }
```

This instruction is useful to in contact with the wall.

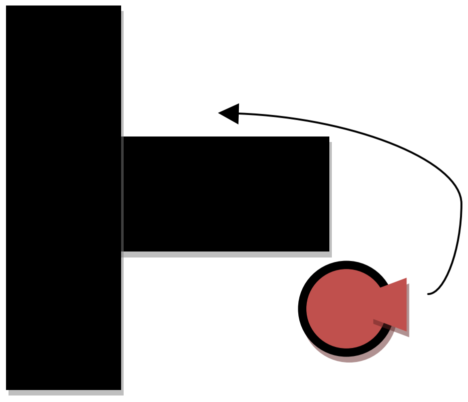

Figure 2

Reactive navigation

```
 if (sensors_value[5]<150 && Wall==1) {
   speed[0]=100.0;
   speed[1]=250.0;
 }
```

In this case the wall is too far so we reduce the speed of the left wheel to come closer. As shown in the above Figure 2.

Bug Algorithm:

Spatial knowledge

We need the X Y position of the robot in order to know where we are and if we reach our goal or not. We don't use the Z coordinate we assume we are in a flat space. We also need to know the direction of our robot

```
double X=0,Y=0;
double Xgoal=0.10,Ygoal=0.10;
```

We set e-puck position and goal position.

```
double dl = l / ENCODER_RESOLUTION * WHEEL_RADIUS; // distance covered
by left wheel in meter
double dr = r / ENCODER_RESOLUTION * WHEEL_RADIUS; // distance covered
by right wheel in meter
double da = (dr - dl) / AXLE_LENGTH;            // delta orientation
if (da<=0){
da=-da;
}
while (da>=6.28){
  da=da-6.28;
}
```

We know the distance covered by each wheel and the orientation of the e-puck. We keep the orientation between 0 and 6.28 (2π)

```
mvt=mvt-prevmvt;
```

This instruction computes the movement done between the previous and actual iteration of the odometry step.

```
X=X+sin(da)*mvt;
Y=Y+cos(da)*mvt;
```

We increment the X and Y position according to the orientation of the robot.

As shown in the Figure 3, we are calculating the shortest path to the goal from the starting point. Then we are moving to achieve the goal when we are encountered with an obstacle we use the collision avoidance and move away from the obstacle and again calculate the shortest path to the goal and start moving towards the goal and if we encounter the obstacle again we move away and again we calculate the shortest path and move towards the goal. In this way we reach our goal.

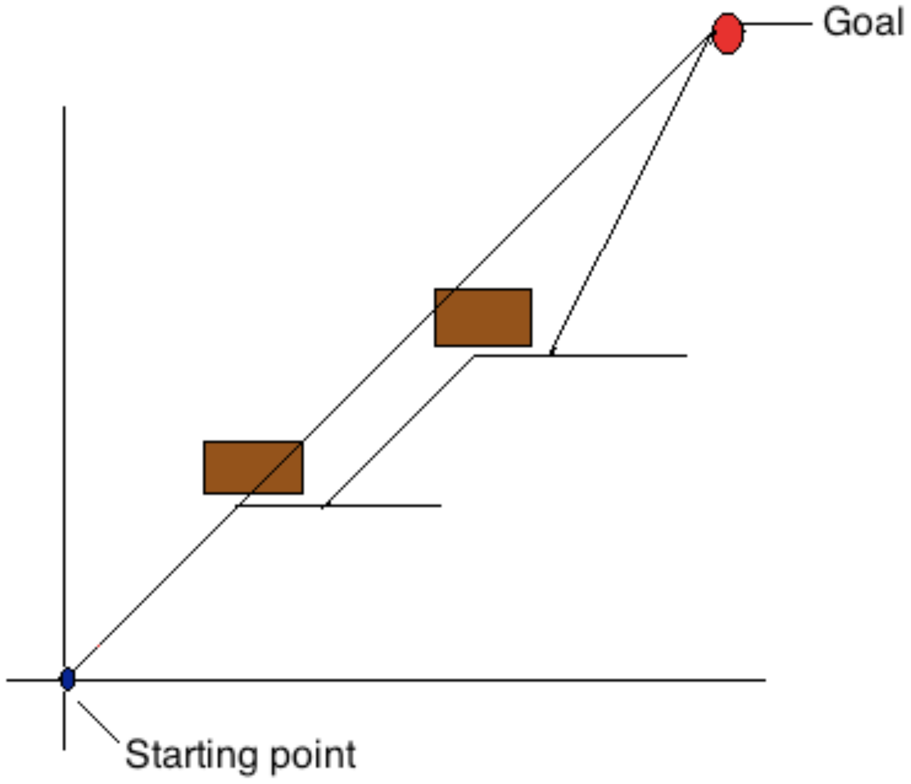

Figure 3

Navigation:

We give priority between instructions. We describe them by importance order
The most important if we reach the goal then stop

```
if (((X>0.98*Xgoal)&&(X<1.02*Xgoal))&&((Y>0.98*Ygoal)&&(Y<1.02*Ygoal))){
  speed[0] = 0.0;
  speed[1] = 0.0;
  printf("goal reach");
}
```

We keep a margin for the position of the goal
We keep the instruction to follow the left wall.
Finally we try to find and follow the shortest path to the goal.