Autonomous Robots

L5. Obstacle Following

This document will guide you through the necessary steps to program an obstacle following behaviour with the e-puck using the Webots environment.

1. IR sensors

The e-puck has 8 infrared sensors pointing towards 8 different directions. The following picture shows their position:

Each IR sensor sends an infrared electromagnetic wave and measures the amount of light reflected by an object in front. The amount of light that the IR sensor measures, depends on the distance between the sensor and the object, but also on the surface of the object and the ambient light. We can have different measures if the environment is illuminated by a fluorescent light or by an incandescent light. We can also have different measures if the object is a white paper, a black paper, a white plastic or a black plastic.

In order to relate the amount of light received by the sensor and the distance between the sensor and an object, it is necessary to calibrate the sensor response. The following figure shows the value of the IR0 sensor in a backward trajectory of the robot. In the beginning, the robot was touching an object. Then, it moved backwards 6cm away at a low speed (linear_speed=-10):

TO DO: Program and test the same experiment. Use the logFile.txt to plot with Matlab the previous figure.
Now we can use the Matlab Fitting tools to approximate the graph with a 6th or 7th degree polynomial:

\[
y = -2.7\times10^{-3}x^7 + 3.7\times10^{-7}x^6 - 2.1\times10^{-9}x^5 + 6.5\times10^{-12}x^4 - 1.1\times10^{-15}x^3 + 1.1\times10^{-18}x^2 - 0.00055x + 0.14
\]

TO DO: Find the polynomial that approximates your data.

TO DO: Include the polynomial in your code in order to convert the 8 IR measures to 8 distances:

```c
double coef[7];
...
coef[6]=2.2372*pow(10,-21);
coef[5]=-2.7966*pow(10,-17);
coef[4]=1.3764*pow(10,-13);
coef[3]=- 3.4013*pow(10,-10);
coef[2]=4.465*pow(10,-07);
coef[1]=- 0.00030517*pow(10,0);
coef[0]=0.10252*pow(10,0);

for (i = 0; i < 8; i++) {
    value = wb_distance_sensor_get_value(distance_sensor[i]);
    sensors_value[i] = 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
}
```

Now you can start building your obstacle following behaviour.

2. Obstacle following behaviour

Write a high-level controller that moves forward until it finds an obstacle. Then circumnavigate forever the obstacle until you lose it. In that case go forward until a new object is found.

Although you can program this behaviour with alternative strategies, it is recommended to program it by considering these 3 following states:

State 0: go forward until IR0, IR1, IR6 or IR7 detect a close obstacle. Then change to state 1.
State 1: turn the robot left until IR2 detects the same close distance. At this point the robot will be perpendicular to the surface and you can then change to state 2.

State 2: move the robot forward at a constant speed (linear_speed=60). Use the information of IR1 and IR2 to set the angular_speed. This speed has to be correctly applied to follow the wall at a correct distance (i.e.: 1 cm) and to turn when a corner is achieved. Change again to state 0, if IR1 and IR2 distances become too high (you loose the object).

TO DO: program and test your obstacle following behaviour.


Write a report explaining in detail the work done in sections 1 and 2. Add some graphs about the trajectory done by the robot with your obstacle following behaviour. Explain also the problems you found.