Comp3302 Navigation by Potential Fields

C.B.Price December 2020

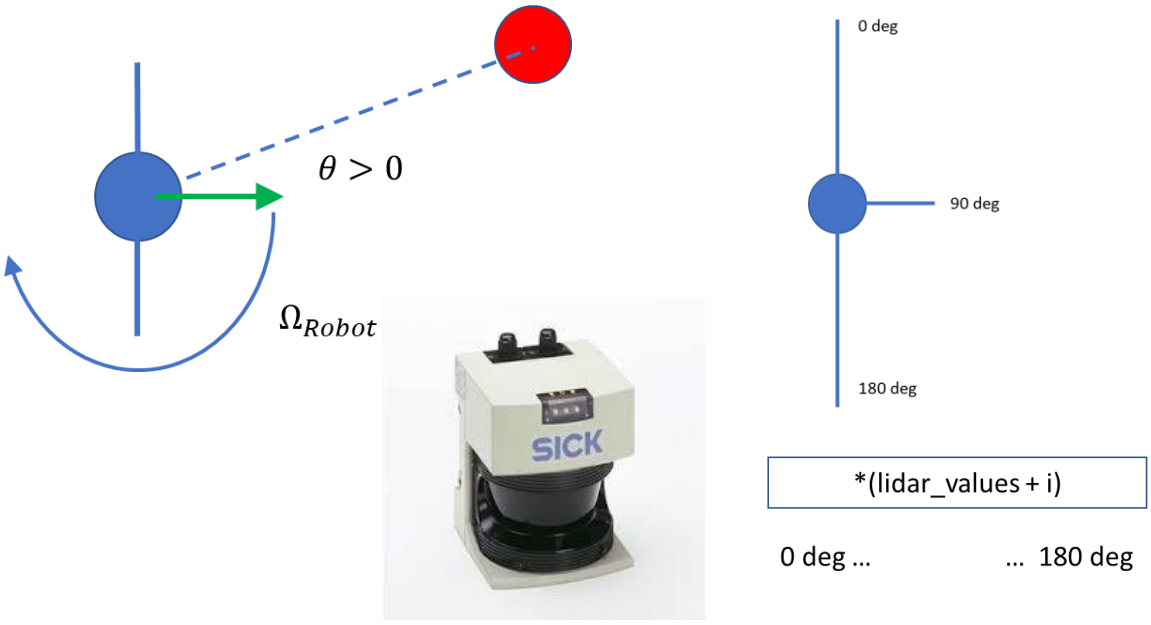
Purpose	To investigate navigation using potential fields (forces)
Files Required	Webots project folders on website. See activities below.
ILO Contribution	PP3
Send to Me	If you are working online, please send movie-clip of your solution.
Homework	

Activities

---

1 Algorithm Idea

True navigation using potential fields involves creating a field of hills centred on each obstacle. As the robot moves, it tends to move down these hills and so avoids the objects. This requires a heavy off-line computation of the field of hills. Here we take an equivalent approach, using forces. When the robot approaches an obstacle, it experiences a force which rotates it away from the obstacle. This is sketched below for an object to the left of the robot which is moving to the right (green arrow)



The robot arena is scanned using a LIDAR (laser-based) scanner (centre photo). This scans from left to right and builds an array **lidar_values** which stores the distance to any obstacle detected at each angle from 0 to 180 degrees (right diagrams).

The controller generates values of **omegaRobot** to make the robot rotate. This comprises two components, one from the obstacles and the other, an attractive Braitenberg-like force towards the light,

omegaRobot = omegaRobotForce + omegaRobotLight;

The robot is also given a steady forward speed **velyRobot** and use this together with **omegaRobot** to drive the wheel motors, like this

omegaR = velyRobot + omegaRobot;
omegaL = velyRobot - omegaRobot;

---

2 Investigate the LIDAR device

Let's see the LIDAR at work. Open the world **CBP_Pioneer3_Sick_World_2.wbt** and select the controller **CBP_Pioneer3_Sick_0.c.**

You will see the LIDAR point-cloud in a separate display. The console shows the LIDAR array of length 180, from 0 to 179 degrees, from top left to bottom right. The arena is 8m x 8m and the Pioneer3 robot is at its centre, facing right.

Make sure 'Show Lidar Point Cloud' is selected in optional rendering.

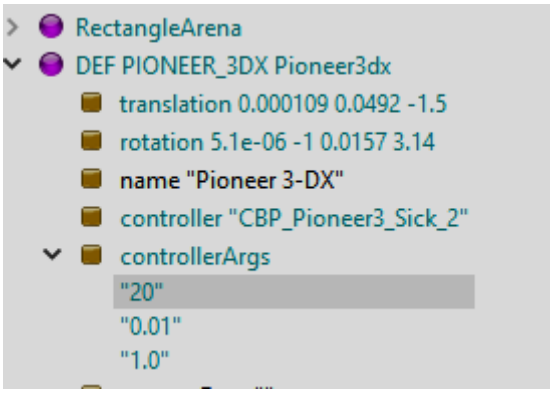
(a) So run the controller and look at the display as you move the obstacle around. Move it left and right, far and near and also behind the LIDAR device (behind the robot)

(b) Repeat but look at the numbers in the console. These will be around 4 metres since the arena is 8m x 8m. Move the obstacle to the 0-degrees LIDAR position and observe the console numbers give the distance to the obstacle.

---

3 Investigating the robot navigation.

(a) Investigate the controller parameters. These are specified in the PIONEER node in the Scene Tree:



These are currently set to k_att = 20 (attraction to light), k_rep = 0.01 (object repulsion) and d_0 = 1.0 (force cutoff). You can change the parameter by clicking on it, then **save** the world.

(i) Increase k_att a few times and see how the robot's behaviour changes
(ii) Increase k_rep a few times and see how the robot's behaviour changes (restore k_att to default)
(iii) Play around with d_0 and find out what this does (restore k_rep to default)

(b) Now add more obstacles and see how well the robot navigates. Try to plan the scenario's you create. You might find there are scenario's which the robot cannot handle.

---

4 Explaining code (required for PP3)

(a) Explain how the function **compute_force_obstacle(…)** works

(b) Let's make an overview of how the robot is made to move. Then you can put this into nice words for your position paper.

```
double leftIntensity  = wb_light_sensor_get_value(leftLight);
double rightIntensity = wb_light_sensor_get_value(rightLight);

omegaRobotLight = k_att*(leftIntensity - rightIntensity)/200;
```

Above lines get the intensity values from the left and right sensor. The difference is used to provide a rotation speed to be used later

```
range = lidar_values[i];
thetaObj += (PI/2 - (double)i*PI/180);
forceRep += compute_force_obstacle(range);
```

*You'll find the above lines in a loop which iterates over **i** (0 – 179). It sums angles of any objects detected and sums repulsion forces. This then calculates the average angle **avTheta** to be used later.*

```
if(avTheta < 0) {
   omegaRobotForce = forceRep*(PI/2 + avTheta);
} else if(avTheta > 0){
   omegaRobotForce = -forceRep*(PI/2 - avTheta);
} else {
  omegaRobotForce = 0;
}
```

The lines above take the average angle to the obstacle, and add or subtract 90 degrees. This generates a force to the side of the obstacle. The size of the force is also proportional to **forceRep** *calculated before. This produces a rotation speed to be used later. Note how the rotation speed calculation is different according to the location of the obstacle, to the left or to the right of the robot direction*

```
omegaRobot = omegaRobotForce + omegaRobotLight;
omegaR = velyRobot + omegaRobot;
omegaL = velyRobot - omegaRobot;
wb_motor_set_velocity(left_wheel, omegaL);
wb_motor_set_velocity(right_wheel, omegaR);
```

Finally we combine rotation speeds from the light attraction and object repulsion. We use these to calculate the motor drive speeds **omegaL omegaR** *and then we pass these to the wheels to make the robot move. Note there is a constant forward velocity component* **velyRobot** *otherwise the robot would not move at all.*