Comp2403 Navigation by Potential Fields

C.B.Price November 2020

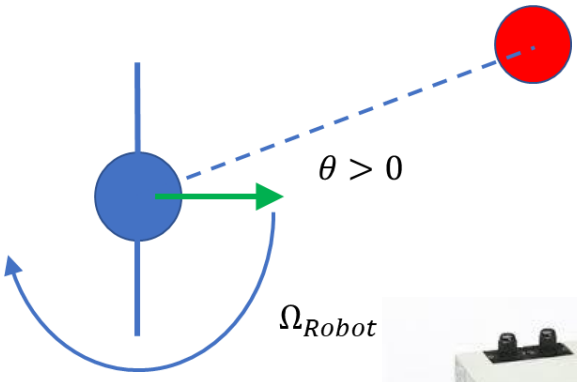
Purpose	To investigate navigation using potential fields (forces)
Files Required	Webots project folders on website. See activities below.
ILO Contribution	1,2,3
Send to Me	If you are working online, please send movie-clip of your solution.
Homework	

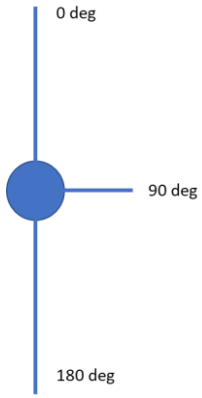
Activities

---

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)





*(lidar_values + i)

0 deg … … 180 deg

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).

We shall program the controller by generating 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;

We shall also give the robot 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

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** which outputs the LIDAR array from 0 to 180 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 LIDAR output. Verify that it is correct as you move the obstacle around. In other words the array of distances is indexed from 0 to 180 degrees, from robot left to robot right.

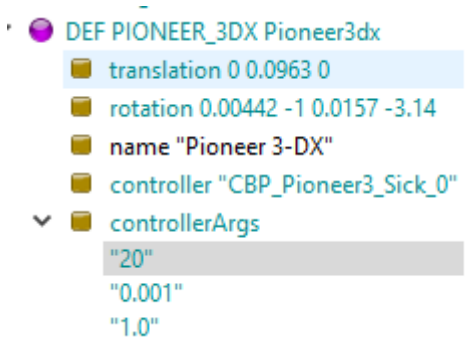
3 Program the Force Navigation Controller (1) Attraction to Target Light

Stay in the same world and select the controller **CBP_Pioneer3_Sick_2.c** First we shall code a Braitenberg-like sensor to rotate the robot towards the light

(a) Look where the left and right eye intensity is grabbed from the sensor, and use these values to complete the line of code starting **omegaRobotLight =** to make the robot rotate towards the light.

Please use the attraction coefficient **k_att** and you may like to divide by **200**, the max value returned by the sensor. You can see this in the **lookupTable** in the robots **extensionSlot** like this

(b) Compile and run your code and make sure that the Pioneer moves towards the light. You may like to experiment with the attraction coefficient which you can set here (no need to change and re-compile the code),



4 Program the Force Navigation Controller (2) Object repulsion

(a) The expression for the repulsive force we shall use is

$$F_{obstacle} = K_{rep}\left(\frac{1}{d} - \frac{1}{d_0}\right)\frac{1}{d^2}$$

for $d < d_0$, else the force is zero.

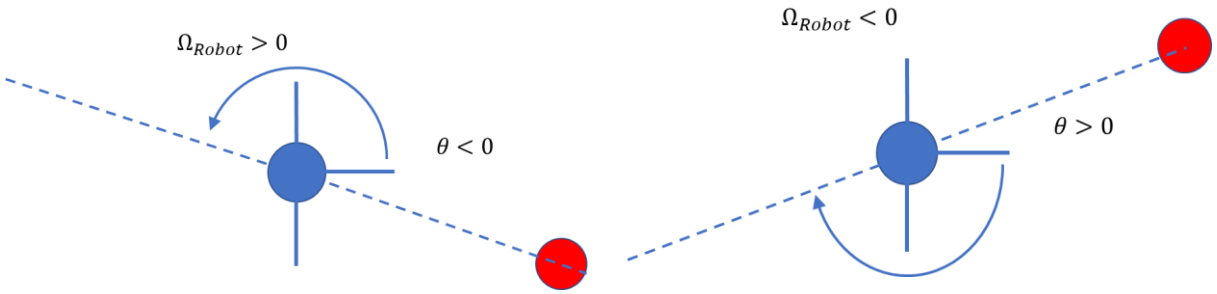
You will work out which variables to use. In the c-language the function to raise variable a to power x is **pow(a,x);** Put your code inside the function **double compute_force_obstacle(double dObj)**

(b) Now we must compute the forces. Code goes after "// Compute forces here -------- "

(i) Get each object **range** from the **lidar_values[]** array and test if this is less than the **RANGE_THRESHOLD**
(ii)If it is then get the current angle **(double)i*PI/180** and subtract this from **PI/2** to get the angle in the range of +PI/2 to -PI/2
(iii) Add this angle to the variable **thetaObj**
(iv) Make a call to the function to compute the force for the current **range** and add it to the **forceRep** variable
(v) increment **count** so we know how many angle and range values have been summed.
(vi) Finally after the loop, use values you have used to calculate **avTheta** the average angle location of the object in space, relative to the robot.

(c) Now we must use the force to set the angular velocity of the robot **omegaRobotForce**. So we need to code an expression using **forceRep, PI and avTheta.** We need **PI** since we want the robot to turn tail on the obstacle.

There are two cases, depending on which side of the robot's forward movement the obstacle is located. Here they are:



On the left you can see the obstacle is to the right of the robot, **avTheta < 0** so the robot must rotate in an anticlockwise way to avoid the obstacle, so **omegaRobotForce** must be positive. The opposite is true for the case where **avTheta > 0.**

So come up with code to do the calculation of **omegaRobotForce** for these two cases.

5 Investigate your controller and the robot behaviour

There's tons to investigate here. The goal is to get a controller which will produce successful navigation in a cluttered arena.

But there are limitations, so don't expect too much. Remember we used the *average* object angle and range. This may not work well for multiple objects. An improved method could be used as in the object-detection worksheet. Perhaps this is work for a mini DBT project.

Some ideas

- You have 3 parameters accessible via **controllerArgs** so you could have a look at changing these (one at a time)
- The effect of location of obstacles
- Does the solution *scale-up*? I mean can it deal with more than a small number of obstacles
- What is the effect of changing the light intensity?
- Can multiple robots move around and avoid each other? Perhaps you could use multiple lights?