Comp2403 Object Detection and Localization

C.B.Price October 2020

Purpose	To learn how to scan the environment to detect obstacles and move to a nearby obstacle
Files Required	Webots project folders on website. Use the world **CBP_ePuck_ObjDet_1.wbt** with the associated controller c-file for a single obstacle, then **CBP_ePuck_ObjDet_2.wbt** with the associated controller c-file for motion to the gap between two obstacles
ILO Contribution	1,2
Send to Me	If you are working online, please send movie-clip of your solution.
Homework	Look at Seigwart & Nourbakhsh Chapter 5 or CBP_Notes Chapter 4

Activities

---

1 Detecting a Single Object

The world **CBP_ePuck_ObjDet_1.wbt** contains our friendly ePuck and a single oil barrel. The controller code makes the following happen:

The robot rotates and scans the arena with its time of flight sensors. At each angle, it stores the distance measured in an array. The index into the array is the angle.

Then the array is analysed: The controller finds the closest point to the robot (minDist) and the angle at which this occurred.

The robot then rotates to this angle.
Then it moves minDist using odometry.

(a) Look for the above behaviours in the code, each is associated with a state, and make sure they make sense. Pay particular attention to the code associated with the robot sensors

(b) In **STATE_ANALYZING** complete the code to iterate over the distArray and find the minimum distance, and the angle (array index) at which this occurs. Compile.

(c) Towards the end of **STATE_ROTATETO_OBJECT,** when the robot has stopped moving, add a single line of code to get the left wheel encoder value. This will be used later

(d) Now let's turn to coding the **STATE_FORWARDTO_OBJECT.** You need to do the following:
(i) Add a line of code to get the changing left wheel encoder value. You must use the previous value calculated in (c) so you measure how far the robot has moved.
(ii) Convert this value to the distance moved at any time
(iii) Now complete the **if( )** block of code to stop the robot when it has travelled the distance

(e) Compile, run and investigate. Fire up Octave and point it to the controller folder. Run the script **Log** which will give you a polar plot of scanned distance values.

---

2 Detecting Two objects and Localizing the Gap between them and moving there.

The world **CBP_ePuck_ObjDet_2.wbt** contains our friendly ePuck and two oil barrels. The controller code makes the following happen:

The robot rotates and scans the arena with its time of flight sensors. At each angle, it stores the distance measured in an array. The index into the array is the angle. There is a second array also indexed by the angle. If the distance is less than a threshold, then the current value of this array is set to 1. This information is then used to locate the centre of the gap between the barrels; the robot rotates to this angle and moves to this point.

Most of the code is present, here we shall add the code required for the algorithm

(a) In **STATE_SCANNING** add the following code where indicated

(i) Convert the float theta into integer thetaIndex then if < 0 add 360
(ii) If dist is less than DIST_THRESHOLD set thetaArray at thetaIndex to 1.0 else to 0.0
(iii) Store the dist in the distArray at this index

(b) In **STATE_ANALYZING** add code to get the index in thetaArray corresponding to a falling edge
You will need to test that thetaArray[i] is 1 and thetaArray[i+1] = 0. You should also test that riseIndex = 0 (maybe).

(c) In the same state, add code to get the index in thetaArray where there is a rising edge.
You will need to test that thetaArray[i] is 0 and thetaArray[i+1] = 1. You should also test that you are to the right of the falling edge, it I > fallIndex

(d) Find the following code and make sure you understand what it is doing

```
double dist1 = distArray[riseIndex+2];
double dist2 = distArray[fallIndex-2];
avDist = (dist1 + dist2)/2.0;
```

(e) Look over the code which calculates the coordinates of the detected obstacle points and also the centre of the gap. Please refer to CBP_Notes.

(f) The rest of the code should already make sense. So compile, run and experiment. There is also an Octave file which should do some plotting.