Comp3302 Sensors

C.B.Price November 2020

Purpose	To understand some ePuck sensors: (i) Time-of-flight sensor, (ii) Infra-red distance Sensors.
Files Required	Webots project folders on website. **CBP_3302_Sensorium_1.wbt** world
PP Contribution	PP3
Send to Me	If you are working online, send me a movie-clip of your solution
Homework	

Activities

---

1 Investigating the time-of-flight sensor.

All the sensors you need for this worksheet have been declared and enabled. This is done in the function deviceInit() which is called from main(). Note how the use of *global* variables avoids passing of a huge number of parameters to this function.

(a) Open the world **CBP_3302_Sensorium_1.wbt** and make sure the controller **CBP_3302_Sensorium_1.c** is selected in the Text Editor.

(b) Within the while loop in the main() function add the following lines to sample the time-of-flight sensor and print out the results

```
double tof_value = wb_distance_sensor_get_value(toflight);
printf("tof value = %f\n",tof_value);
```

(c) To understand what the values mean, here is the LUT for this sensor

```
lookupTable [
  0.00   19.8  0.126
  0.05   58.5  0.032
  0.10  111.0  0.019
  0.20  218.9  0.009
  0.50  531.9  0.007
  1.00 1052.0  0.010
  1.70 1780.5  0.013
  2.00 2000.0  0.000
]
```

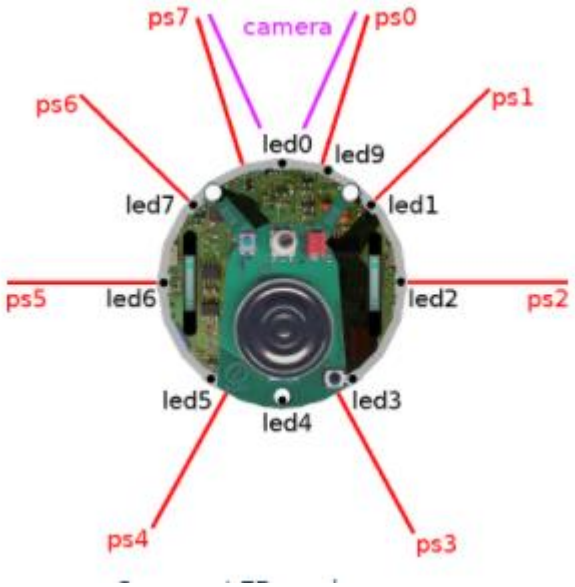
The first column is the actual distance (in meters), the second column is the sensor reading and the third column estimates the 'noise' in the sensor. You should see that the output of the sensor is almost proportional to the actual distance.

(d) Run the simulation. The robot will not move, and the while loop has been slowed down to half a second #define TIME_STEP 500, but you can change that.
Now move OBSTACLE3 and note the tof value change. To understand the values, remember

that the Arena has size 2m x 2m.

2 Investigating the infra-red distance sensors

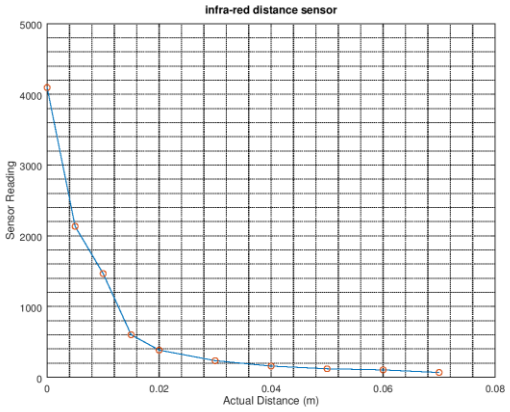
Here's the layout of the 8 infra-red distance sensors plus their names declared in ePuck's PROTO script. Relative to the front of the robot (top in the diagram) they are located at ps0 10 degrees, ps1 45 degrees, ps2 90 degrees, ps3 150 degrees. You can work out the other negative angles by symmetry



(a) Add the following code to the while() loop to sample all of the sensors

```
for(int i=0;i<8;i++) {
  ps_vals[i] = wb_distance_sensor_get_value(ps[i]);
  printf("ps[%d] = %f  ",i,ps_vals[i]);
}
printf("\n");
```

(b) Each sensor has the input-output relationship shown below (expressed in code as a LUT). A larger plot is given at the end of the worksheet.



The first thing to notice is that the sensors have a limited range (0 to 7cms), so they are best

suited for collision detection, not obstacle detection. The second thing to notice is that as the distance to obstacle increases, the sensor reading decreases. Finally, when no obstacle is detected, the 'background' sensor reading is around 67.

(c) Compile and run your code. Verify that all sensors give a reading in the upper 60's when no obstacle is in range.

(d) Now grab OBSTACLE3 and move it towards ePuck's sensor **ps0**. As you approach the sensor, you should see the sensor reading rise.

(e) Explore the other sensor readings by moving OBSTACLE3 around.

3 Inverting the distance sensor LUT

It is inconvenient to work with the raw infra-red sensor output values. It would be better to convert these to actual distances. This is possible, since the sensor works by a LUT which converts actual distances to sensor readings. So all we need to do is to create an 'inverse-LUT' which converts sensor readings to distances.

This is done using the function **lookupDistance(…)** which is in CBP_3302_Helper_1.c in the folder for this controller.

(a) Add the following line of code to calculate the distance from a single sensor. I chose **ps0**

```
double dist = lookUpDistance(c,ps_vals[0]);
printf("dist = %f\n",dist);
```

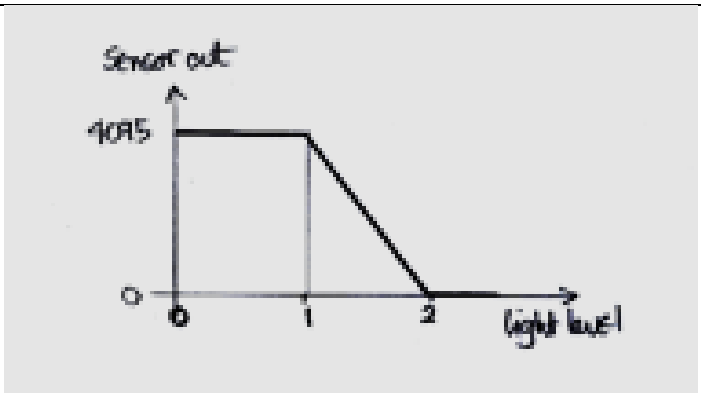
(b) Now experiment moving OBSTACLE3 around and check out that the distances are reported correctly. (I have coded the function so it returns -1 when the obstacle is out of range, further than 0.07m = 7cm away from the sensor).

4 Investigating the Light Sensors

(a) These sensors are located at the same positions as the distance sensors (they share the same electronic devices. Add the following code to sample the light sensors

```
for(int i=0;i<8;i++) {
  ls_vals[i] = wb_light_sensor_get_value(ls[i]);
  printf("ls[%d] = %f  ",i,ls_vals[i]);
}
printf("\n");
```

(b) The relationship between light intensity and sensor reading is given by the following graph:



So a low light level gives a large sensor output value and a high light level a low sensor output value. A function to invert this is provided in CBP_3302_Helper1.c

Add the following code to invert the value of sensor 0:

double output = invertLightSensorVals(ls_vals[0]);
printf("ls0 vals inverted = %f\n",output);

(c) Compile and run. Select **DEF LAMP Solid** in the Scene Tree, and move the LAMP using its gizmo's and observe the light sensor readings, and the single inverted reading.