Comp2403 ePuck Sensors -1-

C.B.Price September 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_ePuck_Sensorium_1.wbt** world
ILO Contribution	1
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_ePuck_Sensorium_1.wbt and make sure the controller CBP_ePuck_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
]
```

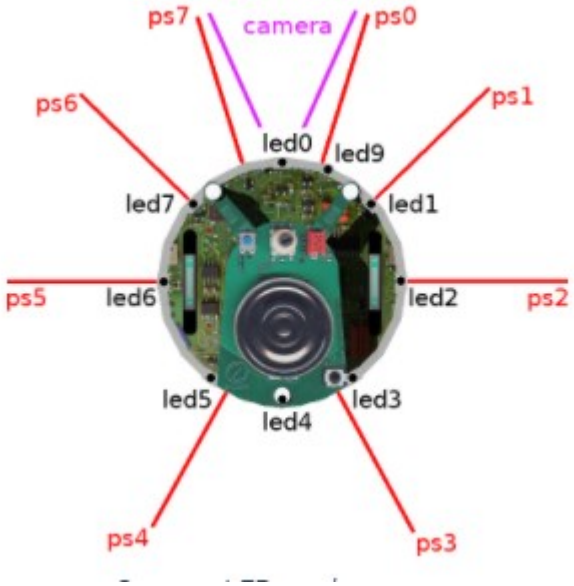
The first column is the actual distance (in meters), the second column is the sensor reading and the third column estimates the 'noise' in the sensor. You should see that the output of the sensor is almost proportional to the actual distance. If you want to plot this out, open up Octave and run the script **plot_LUTs** and select **time-of-flight**. The datasheet for the sensor is included at the end of the worksheet.

(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. You can compare this with the actual distance by opening up the OBSTACLE3 node in the Scene Tree and looking at the **translation** field z-component. You will see this agrees roughly with the tof value. The difference is due to the offset of the tof sensor on the ePuck and the thickness of OBSTACLE3 and of course the LUT! But working all that out is beyond the scope of our work.

2 Investigating the infra-red distance sensors

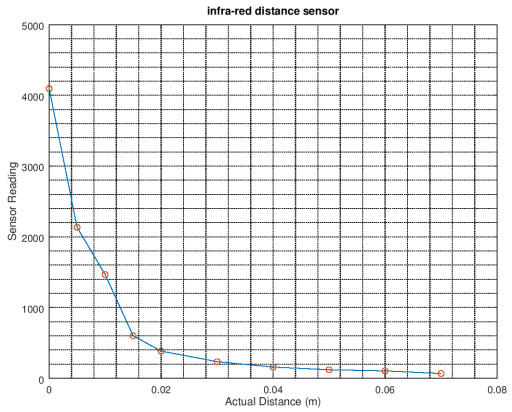
Here's the layout of the 8 infra-red distance sensors plus their names declared in ePuck's PROTO script. Relative to the front of the robot (top in the diagram) they are located at ps0 10 degrees, ps1 45 degrees, ps2 90 degrees, ps3 150 degrees. You can work out the other negative angles by symmetry



(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.



Here is the LUT for this sensor

```
lookupTable [
  0       4095      0.002
  0.005 2133.33   0.003
  0.01   1465.73   0.007
  0.015  601.46    0.0406
  0.02    383.84    0.01472
  0.03    234.93    0.0241
  0.04    158.03    0.0287
  0.05    120         0.04225
  0.06    104.09    0.03065
  0.07    67.19      0.04897
]
```

If you want to plot this out, open up Octave and run the script **plot_LUTs** and select **distance sensor**. The first column is the distance to the obstacle, the second column is the sensor output and the third column is the error.

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 sensor LUT

It is inconvenient to work with the raw infra-red sensor output values. It would be better to

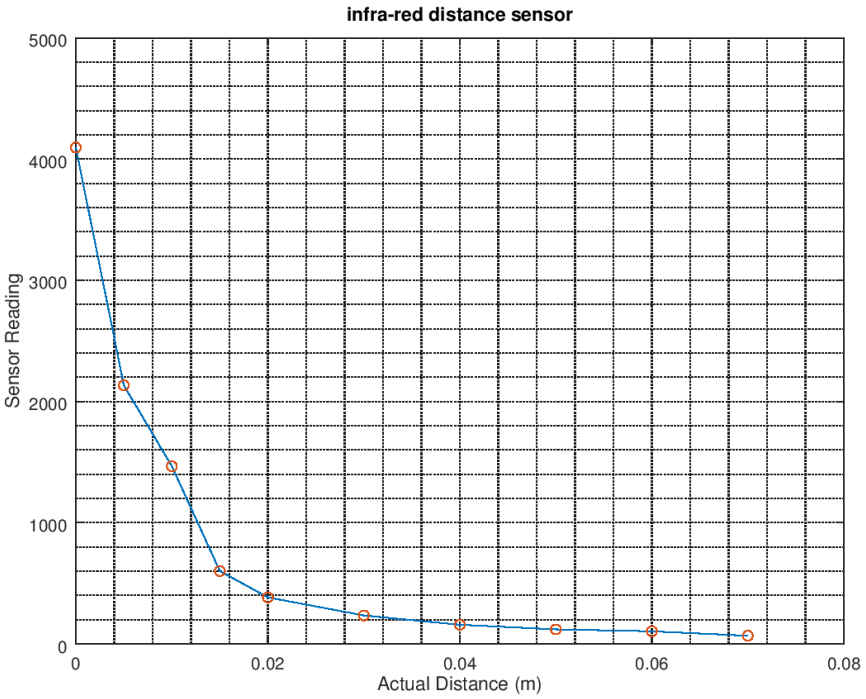
convert these to actual distances. This is possible, since the sensor works by a LUT which converts actual distances to sensor readings. So all we need to do is to create an 'inverse-LUT' which converts sensor readings to distances.

This is done using the function **lookupDistance(…)** which is in CBP_Helper_2.c in the folder for this controller.

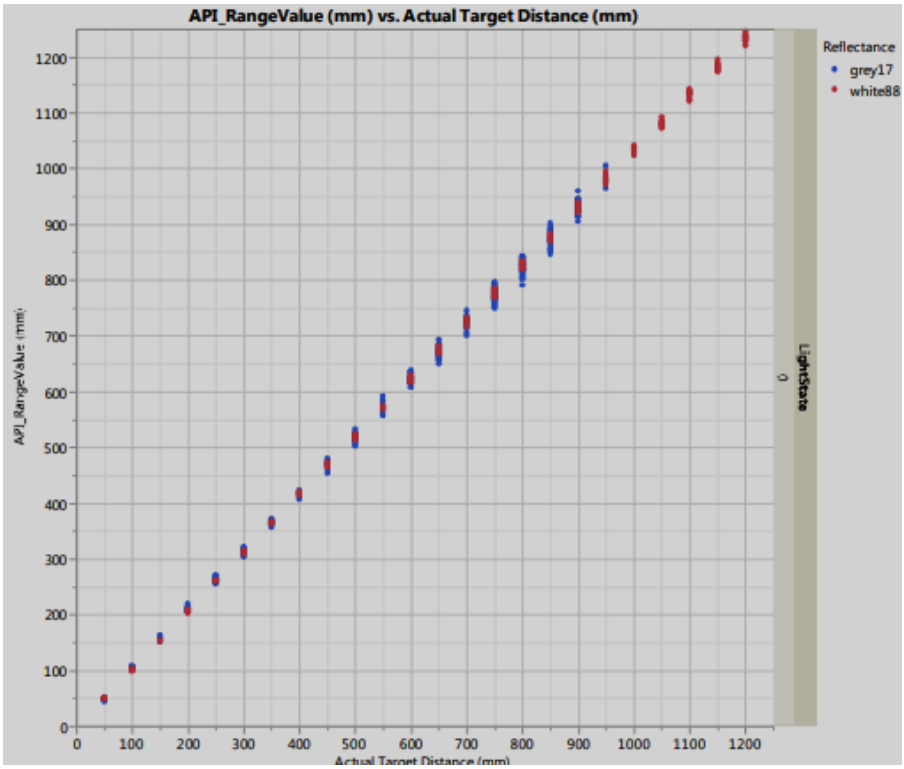
(a) Add the following line of code to calculate the distance from a single sensor. I chose **ps0**

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).



Infra-red distance sensor (above).



Time of flight sensor (above)