Lab 7: Solution

1. Are the sensor measurements changing all the time (see the DistanceSensor tab for example)? Why? Is it the same on a real system?

The measurements might be changing all the time because of sensor noise. In Webots, the noise is simulated by adding a computer-generated random number. On the real e-puck, the same thing happens because of random fluctuations due to physical properties of the sensor and the signal.

2. By moving objects around the e-puck, sketch the response of the proximity sensor as a function of the distance to an obstacle. Note that the range of the e-puck's distance sensors is fairly short.

You will notice that the value is about 60 for far away obstacles. Starting from about 1 robot diameter (7 cm), it increases up to about 4000 (can be lower, depending on the color of the obstacle).

3. What happens if an obstacle and the robot are interpenetrating? What are the proximity sensor measurements in this case?

In that case, the sensor measurement drops to the nominal value of 60 because Webots calculates the distance to the next obstacle, which is in that particular case the wall located on the other side of the obstacle.

4. Examine the values in the control window. Identify which color represents each of the axis of the accelerometer.

The horizontal plane corresponds to XY, the vertical axis corresponds to Z.

6. Open the supervisor controller. Try to understand what the code is doing. Now, compile and build this code and run the simulation.

It is creating two sin signals with varying amplitudes using FREQ_0 = 0.25 and FREQ_1 = 2.0 and assigns them to each of the two light sources. FREQ_0 and FREQ_1 are defined in lines 7 and 8 of file lights_supervisor.c of the light_supervisor controller. The values of the sin signals change according to what is implemented in lines 32 and 33 of the same file.

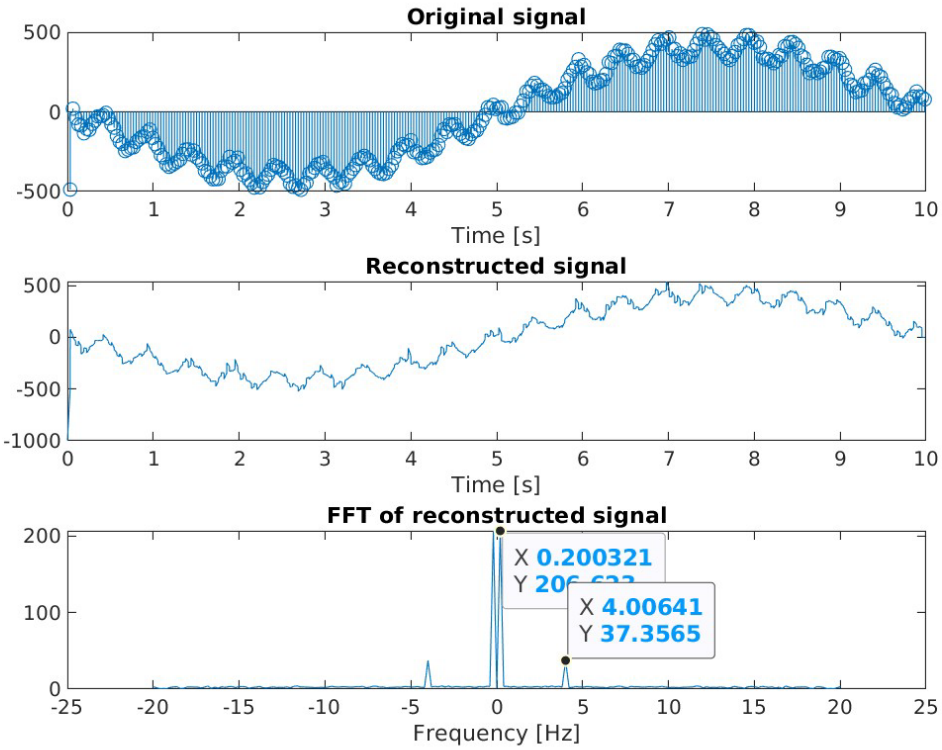
8. Take a look at the FFT of the interpolated signal. How many frequency components do you observe?

The red light is turning on and off faster than the white light (it is pulsing at a higher frequency). The plot in time domain shows the combined effect from the fast oscillations generated by the red light and the slow oscillations generated by the white light.

To compute the FFT of the signal, it is necessary to note the sampling frequency of the signal. This frequency can be achieved by looking at the difference between two time stamps of the first column of the variable ls_values. You can confirm that this time difference is 0.032 seconds, which corresponds to the TIME_STEP declared in the e-puck_ls.c file of the e-puck_ls controller.

Finally, In the FFT of the reconstructed signal you can clearly observe peaks at 0.25 and 2 Hz which correspond to the different lights. Note that to compute the FFT of the signal

Note that the precise graph observed depends on the position of the robot, lights and obstacles.



9. Run the simulation without changing the controller in *e-puck_obstacle.c*. How is the e-puck robot behaving? Is the robot avoiding the obstacles? Go to *e-puck_obstacle.c* file and identify the lines of code responsible for setting the wheel speeds of the robot.

In the file e-puck_obstacle.c, the lines 37 to 40 are the ones responsibe to set the wheel speeds of the robot. Since this code only place both wheel speeds at the same velocity, the robot is doomed to go forward without avoiding any obstacle.

10. In that part of the code, implement a controller for the wheel speeds of the robot, based on a set of simple rules (if -> then -> else) on the values of the proximity sensors, so that the robot can avoid obstacles.

```
//set the speed of each wheel
speed[0] = 500; //standard behavior (no obstacles detected)
speed[1] = 500;
duration = 20;
for (i = 0; i < NB_SENSORS; i++){ //go through all sensors
    if( (i != 3) && (i != 4)){ //exclude back sensors
        if(ds_value[i] > 500){ //if any sensor detects an obstacle
          speed[0] = 300; //turn in place for a while (1 second)
          speed[1] = -300; //note that standard behavior changes
          duration = 1000;
          break;
        }
    }
```

```
}
```

11. Which vehicle depicted do you expect to be most effective at avoiding obstacles if light sensors were to be replaced by proximity sensors, and why?

The vehicle 3a is the most efficient one since it will basically slow down the left wheel upon detection of an obstacle on the right, thus making the robot gracefully avoid the obstacle by turning on the left (and vice versa). Vehicle 2a would work as well, but would tend to accelerate when surrounded by obstacles, thus leading to possible collisions, if not carefully tuned.

12. Implement the Braitenberg controller.

```
for (i = 0; i < 2; i++) {
      for (j = 0; j < 8; j++)
        speed[i] += braitenberg_coefficients[j][i] * (1.0 -
  (ds_value[j] / RANGE));

    }
```

18. What is the influence of the parameter, say, $\alpha_{\text{left},0}$ on the speed of the left wheel? *Hint:* explain what happens if $\alpha_{\text{left},0}$ is large or small when an obstacle is detected by the proximity sensor `ps0`.

The parameter $\alpha_{left,0}$ describes the importance of the sensor 0 in the control of the left wheel. Namely, if the parameter is large, the left wheel will slow down a lot when an obstacle is detected by ps0. If the parameter is small, the left wheel will not be much affected by the detection of an obstacle by ps0.

19. Modifiy the parameters of the controller *e-puck_braitenberg.c* so that you achieve these behaviors:

1. The robot is moving forward while smoothly avoiding obstacles.

```
double braitenberg_coefficients[8][2] =
  { {140, -35}, {110, -15}, {80, -10}, {-10, -10},
  {-15, -10}, {-5, 80}, {-30, 90}, {-20, 160} };
```

2. The robot is moving forward while being attracted by obstacles.

You will obtain a close-to-perfect behavior simply by inverting the weight above, i.e., by assigning the weights of the left wheel to the right wheel, and vice versa.