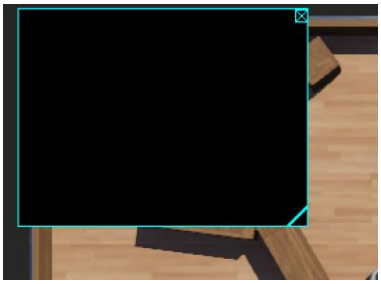
LAB 12 – Improved Sensor Model Mapping

(1) Download the **Lab12_ImprovedSensorModelMapping.zip** file and unzip it. Load up the **CratesWorld** world. The world should appear as shown below. You may have to turn on **3D View** from the **Tools** menu.

The goal of this lab is to apply an **improved** sensor model (i.e., using Gaussian distribution) to each reading as we add it to the occupancy grid. This will allow more certainty around the center of the reading and less certainty as the reading spans outwards. We will also eliminate some noise by reducing the certainty for all areas inside the reading wedge that appear before the object as well as zeroing out the area underneath the robot, as we know that there are no objects under the robot.

The code will make use of the code from **Lab 11** (i.e., the **Sensor Model** lab). You will make use of a revised **MapperApp**, which allows for additional grayscale coloring of the occupancy grid and floating point values in the grid as well as the use of a temporary grid so as to reduce inconsistencies when applying the readings. As with the previous lab, there will be two additional files in the controller directory (**MapperApp**.java and **Map**.java). You will again be altering **Map.java**.

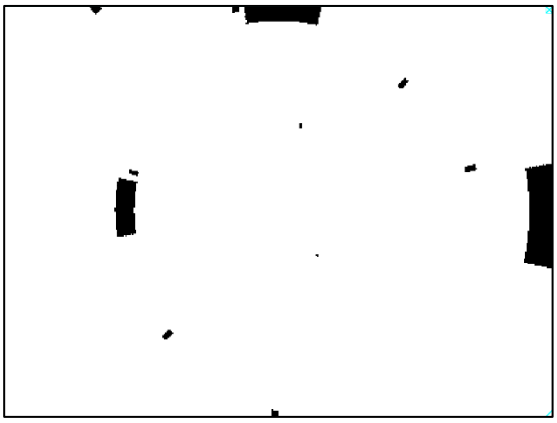
IMPORTANT:



This code makes use of a display window that will appear as black upon startup. The display window will show the map. It can be resized by grabbing the bottom right corner. You should NOT ever close it using the top right corner. If you close it accidentally, you can re-open it by clicking on the robot … then, from the **Overlays** menu select **Display Devices** then **Show 'display' overlay**. If this doesn't work, you can delete the WBPROJ file in the worlds directory that corresponds to the world that you are using, then re-open or reload the world. However, the WBPROJ files are hidden files, so you need to enable your windows/mac to show hidden files. On Windows pcs, check of **Hidden Items** under the **View** menu of the **File Explorer**. On a Mac, press **control+shift+period** to show hidden files.

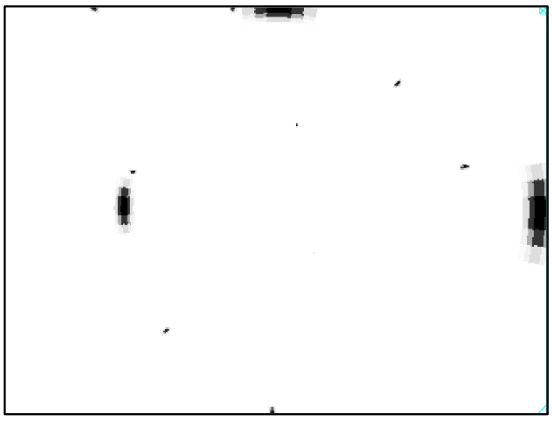
In the previous lab, we applied the sensor readings directly to the occupancy grid. We will now apply all the readings from a single robot position (i.e., **9** IR and **5** sonar) to a temporary grid and then call **updateGrid(x, y)** in the **MapperApp** class to transfer these temp changes onto the main grid map. You will notice the call to **mapper.updateGrid(x, y);** at the end of the **addSensorReadingsToMap()** function in the **Lab12Controller** class.

(2) The controller code starts with the **MAX_SPEED** of the robot set to **0**. Also, the first two lines from the **Map** class's **draw()** function have been commented out and the code at the end of the **while** loop in the controller will break after one iteration. The **addSensorReadingsToMap()** has been set so that it sends all IR and SONAR readings that are less than **120cm**. Run the code. You should see something like what is shown here on the right →



Adjust the **applySensorModelReading()** function in the **Map.java** file so that it applies a Gaussian distribution across the sensor model readings. Currently, the code does this to set the values: `setTempGridValue(x, y, 1);`

This **tempGrid** has been defined for you as an attribute in the **Map** class and this is the function that sets the grid value for a single cell in the map. However, it only sets the value to **1** … representing a 100% probability that the object is there. That is why the map above shows a "uniform" blotch of black for each reading. Instead of setting to **1**, you need to put in the sigma probability which is one of the **36** values shown on slide 16 from the notes. You can make an array and simply multiply the distance probability times the angular probability to get the correct overall probability. All that is needed is the indices into the array … the **angleIndex** and **distanceIndex**. These have been given for you in the code and you should make use of them. The **updateGrid()** method has been completed for you so that it transfers (i.e., by adding) the values from the **tempGrid** onto the **occupancyGrid** while maintaining the maximum and minimum values needed for display purposes. Once you are done, when you run your code you should see something like what is shown here → Save a screen snapshot of the **MapperApp** window as **Snapshot1.png**.



(3) If all the above is working fine, you should be pretty sure that your application of the sensor models is working properly. It is time to start producing some maps. Put back the first two lines in the Map class's **draw()** function:

```
if (DISPLAY_COUNTER++ %250 != 0)
   return;
```

Comment out this line at the end of the **while** loop in **Lab12Controller.java**:

```
if (count++ == 1) break;
```

Set the **MAX_SPEED** of the robot to **15**. With the sensor readings still set to be mapped at distances < **120cm**, run the simulation until the timer indicates 0:**05**:00:00**.** Save a screen snapshot of the **MapperApp** window as **Snapshot2.png**. You will notice that this is one ugly map! That is because we are allowing long range readings with a large error.

Reduce the sensor mapping distances to **30cm** in the **Lab12Controller** code and re-run everything again … but this time until the timer indicates 0:**10**:00:00 so that we get a darker image. Save a screen snapshot of the **MapperApp** window as **Snapshot3.png**.

Reduce the sensor mapping distances even further to **10cm** now in the **Lab12Controller** code and re-run everything again until the timer indicates 0:**10**:00:00. Save a screen snapshot of the **MapperApp** window as **Snapshot4.png**.

(4) Set the sensor mapping distances back to **30cm** in the **Lab12Controller** code. Write code in the **applySensorModelReading()** method of the **Map** class that decreases the likelihood of an obstacle all along the path leading up to the Gaussian distribution readings that you used. Slides 19 and 20 explain this. Just use **setTempGridValue()** with a negative value for the cell based only on the angular probability.

Make sure to apply the angular distribution as shown in slide 19 of the notes so that the center of the wedge is more certain of no obstacle than the outer parts of the wedge, according to the 6-sigma rule (note that we are not applying the probability along the distance … so you will not be multiplying the distance probability with the angular probability for that portion from the robot up to the distance reading range).

Run the simulation until the timer indicates 0:**10**:00:00. Save a screen snapshot of the **MapperApp** window as **Snapshot5.png**. Compare this image with **Snapshot3.png**. What do you notice? You should see that the edges are more well-defined as lots of the uncertainty has been removed. Unpause the simulation to let it run until the timer indicates **0:30:00:00.** Save a screen snapshot of the **MapperApp** window as **Snapshot6.png**.

(5) Now we will clear the area under the robot at the robot's location so that with 100% certainty, there is no obstacle under the robot. Uncomment the call to **clearAroundRobot(rx, ry);** at the bottom of the **updateGrid()** function in the **Map** class. For all occupancy grid cells covered by the robot's body at that moment (i.e., based on the robot's radius), the code sets their value to **0** to indicate no obstacle is there. Run the simulation (again with the **30cm** distances) and let it run until the timer indicates 0:**30**:00:00**.** Save a screen snapshot of the **MapperApp** window as **Snapshot7.png**. What do you notice when compared to **Snapshot6.png**? You should see some of the corners made more precise as well as some of the bumps along the edges. Basically, wherever the robot travelled, it should be more refined.

(6) Now load up the **StillMessyRoom** world. Set your code to use ALL sensor readings from both types of sensors but only if they have a distance of less than **20cm**. In the 3rd line of the **addSensorReadingsToMap()** method, change the value of **y** to: -(values[2]*100) **+15**;

Run your code until the timer indicates 0:**10**:00:00. Save a screen snapshot of the **MapperApp** window as **Snapshot8.png**. Compare it with **Snapshot13.png** from **Lab 11** shown here→

In your results, you will notice that there is a lot less noise now … which was from the inaccurate IR readings along the shiny walls and there are no more curved sonar readings. Of course, the tradeoff is that the edges are a little less defined (since lots of sensor readings were corrected/erased).

(7) Remove the **+15** that you added to **y** in the **addSensorReadingsToMap()** method. Load up the **AlignedCratesWorld** world. Try to make the best map that you can within the simulation time of 0:**10**:00:00. To do this … pause the simulation once in a while and translate the robot around so that it heads towards an unmapped wall. Then unpause. You can do this many times within the **10** simulation minutes. Keep in mind that the map only updates every **250** cycles so it may take time for your newly mapped wall results to appear. Save a screen snapshot of the **MapperApp** window as **Snapshot9.png**. Load up the **CombinedCratesWorld** world as well and try to make the best map that you can within the simulation time of 0:**20**:00:00. Save a screen snapshot of the **MapperApp** window as **Snapshot10.png**.

Debugging Tips:

- If you get an **ArrayIndexOutOfBounds** Exception, then likely you forgot to remove the +15 that you added to **y** in the **addSensorReadingsToMap()** method.

Submit your **Lab12Controller.java** code along with the two **MapperApp** files as well as the **10** snapshot **.png** files. Make sure that your name and student number is in the first comment line of your code.