

LAB 10 – Mapping

- (1) Download the **Lab10_Mapping.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 understand how a robot can create a simple map of the environment by using infrared distance sensors. The code will make use of the code from **Lab 1** (i.e., the **BasicMovementAndSensing** lab) as well as your solutions from **Lab 8** (i.e., the **OdometryCorrection** lab).

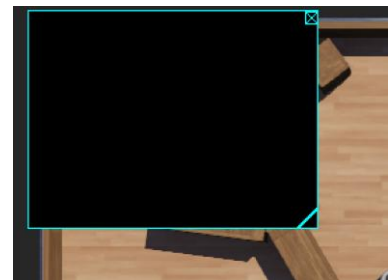
You will make use of the **MapperApp**, which makes use of a simple internal webots display for showing the robot's map. There will be two additional files in the controller directory (**MapperApp.java** and **Map.java**). These will automatically be compiled and used by your program. You should NOT change these files.

You will be making quite a few maps. Once you get the code working, you and your partner should share the code and you can each share in making the different maps (at the same time so that you save time).

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 '**e-Puck Overlays**' then **Display Devices** then **Show 'display' overlay**. If the window does not appear, check to make

sure that **Hide All Display Overlays** is unchecked in the **Overlays** menu. Sometimes you have to reload the environment and sometimes even restart Webots to make the display appear. 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.

- (2) The **Lab10Controller** code has been started for you. It causes the robot to roam around in the environment avoiding obstacles as in **Lab 1**. The robot starts at position shown above $(x_0, y_0, a_0) = (0, 0, 90^\circ)$. Note that the origin is the center of the environment. The program also includes code that computes the robot's position using forward kinematics equations and a compass. Your job is to add code to show a map being built as the robot moves around. Code has been given to open a display window for the map.

A method called **addSensorReadingsToMap()** has been created for you. You need to write code in this function to read all **9** sensors, compute the object location from each sensor reading and display these computed points on the map. The method takes the current (x, y, a) pose of the robot, from which you will take the sensor readings. You should apply the formulas on slide 19 of the notes to get the object position for each sensor. The **x,y** and **angle** offsets of all **9** sensors (see slides 14 and 15) have already been coded for you into arrays ... so that you don't make mistakes.

For now, we will ignore our computation of an estimated position and we will use the actual position of the robot which is obtained from the first three lines of the method.

Once you write that code, you need to insert code in the main WHILE loop so that the method is called repeatedly as the robot moves. You **SHOULD NOT** call the method while the robot is spinning left or right. You should only do it when the robot is in one of the other modes.

Once you are convinced that your code is working, you will create a few maps. Make sure that the **MAX_SPEED** is at **10**. Stretch the display window so that it perfectly overlays the world that you have loaded.

Debugging Tips:

- If your code looks like a random set of lines like this ... then you are likely forgetting to convert your angles to radians in your calls to **Math.sin()** and **Math.cos()**. You can use **Math.toRadians(angle)** to convert an **angle** to radians.



- If your code looks like it is tracing out the robot's path, then you likely forgot to convert to cm by multiplying the readings by **100** when you get the sensor values.



- If your code looks like it is shifting the objects like this ... then you are using the estimated position by mistake instead of the actual position.



- (a) Let your code run for about **5 simulated minutes** on fast forward (until **0:05:00:00** simulation time) then stop the simulation. It should look something like what is below (i.e., white dots outlining the obstacle boundaries).



Save a snapshot of this **MapperApp** window as **ActualDFull.png**. You will notice lots of noisy readings but you should clearly see the shape of most the obstacles ... although some obstacles may not be mapped ... it depends on where the robot ended up travelling. If you see that the robot is kinda re-mapping the same areas over and over, you can always try running fresh again to get a better map.

- (b) Adjust your code so that only distance readings less than **30cm** are recorded on the map. Let your code run for about **5 simulated minutes** on fast forward (until **0:05:00:00** simulation time) and then save a snapshot of the **MapperApp** window as **ActualD30.png**. You will notice less noisy readings and a more defined shape of the obstacles. Likely, since you have eliminated many readings, you will not see the shapes of all the obstacles and you may have even missed portions of the environment, because the robot did not run long enough to reach that part of the environment.
- (c) One more time ... adjust your code so that only distance readings less than **10cm** are recorded on the map. Let your code run for about **10 simulated minutes** on fast forward (until **0:10:00:00** simulation time ... double the time) and then save a snapshot of the **MapperApp** window as **ActualD10.png**. You will notice a lot less noise and see nice clear obstacle boundaries but you may also notice that some areas of the environment are not covered as much. What does this tell you about the accuracy of sensors as objects are further away? No need to provide an answer ... just think about it.
- (3)** Now we will use our estimated positions instead of actual positions. However, you will still use the compass to get the angle of the robot at all times ... so you will not be computing the angle in your calculations. Set the **MAX_SPEED** to **1**. Comment out the first three lines of the **addSensorReadingsToMap()** method. The code for computing the estimates is there already. Change your code to ensure that all

readings are shown again (i.e., not just those less than 10cm). You and your partner should split up the running of the following tests to speed things up:

- (a) Let your code run for about **20 simulated minutes** (until **0:20:00:00** simulation time) and then save a snapshot of the **MapperApp** window as **EstimateDFull.png**. You will notice that the map is similar-looking but there should be some “shifted edges” along some parts of some obstacles due to the robot’s odometry getting skewed. If the edges are shifted too much (see third debugging tip in part 2) then you forgot to set the robot speed to **1**... because the simulator has problems doing proper position estimation at any other speeds.
- (b) Adjust your code so that only distance readings less than **30cm** are recorded on the map. Let your code run for about **20 simulated minutes** on fast forward (until **0:20:00:00** simulation time) and then save a snapshot of the **MapperApp** window as **EstimateD30.png**. You will notice less noisy readings and shifted obstacle boundaries as well. You should see some more doubled edges and again you likely will not see the whole environment coverage.
- (c) One more time ... adjust your code so that only distance readings less than **10cm** are recorded on the map. Let your code run for about **20 simulated minutes** on fast forward (until **0:20:00:00** simulation time) and then save a snapshot of the **MapperApp** window as **EstimateD10.png**. You will notice some crisp edges on the obstacles but you should still see some shifting of the edges.
- (d) Adjust your code to use a **calculated angle** based on the forward kinematics instead of using the compass. You will need to add this appropriate code from **Lab 8**:

```
A = A + ((RightReading - LeftReading) * WHEEL_RADIUS /  
         WHEEL_BASE / Math.PI * 180);  
if (A > 180) A = A - 360;  
if (A < -180) A = 360 + A;
```

Set your code to only record distance readings less than **30cm**. Let your code run for about **20 simulated minutes** on fast forward (until **0:20:00:00** simulation time) and then save a snapshot of the **MapperApp** window as **NoCompassD30.png**. Compare your result with that from **EstimateD30.png**. What do you notice is now the big difference in the map? You should see some double lines on some obstacle borders and perhaps their angles will be skewed. Do you understand how valuable an accurate compass can be when doing mapping? No need to submit answers to these questions ... just think about it.

- (4) Load up the **CratesWithGaps** world provided. Set your code to use the actual position again (instead of the estimate) by uncommenting the first three lines of the **addSensorReadingsToMap()** method and using the compass reading instead of the calculated angle. Set the method to record ALL readings, regardless of the distance. Set the **MAX_SPEED** to **19**. Let the code run for **10 simulated minutes** on fast forward (until **0:10:00:00** simulation time) and submit the snapshot as **Gaps.png**. There is no need to hand in answers to the following questions ... but see if you can figure it out. What do you notice about what happens at the gaps? Some readings successfully compute ranges through the gaps and to the outer

boundary. But why does there seem to be some kind of shape in-between the boxes and the outer boundary?

- (5) Load up the **MessyRoom** world provided. Keep the `MAX_SPEED` at **19**. Make sure that your code is still using the actual position (instead of the estimate) and that you have set it to only map readings less than **10cm**. For this world, the map will be offset by a margin across the top ... I'm not sure why this happens. Just change the third line in the `addSensorReadingsToMap()` method to:
`y = -(values[2]*100) + 15;` and it should be ok. Let the code run for **10 simulated minutes** on fast forward (until **0:10:00:00** simulation time) and submit the snapshot as **MessyRoom.png**. There is no need to hand in answers to the following questions ... but see if you can figure it out. What do you notice about the map? What happens at the mirror? Why are the outside walls not well defined? What is with all that uncertainty on the right side of the map?

Submit ALL of the code you have, as well as the **9** snapshot files. **DO NOT ZIP anything.** Make sure that your name and student number is in the first comment line of your code.