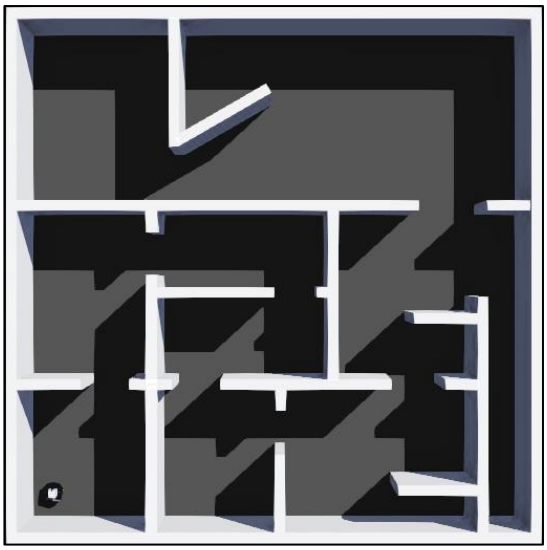
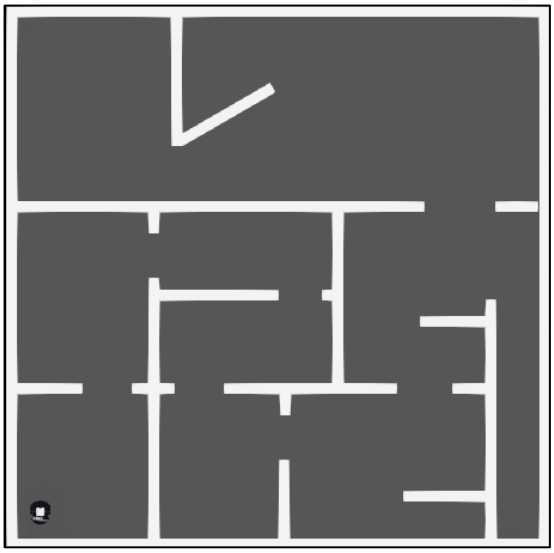
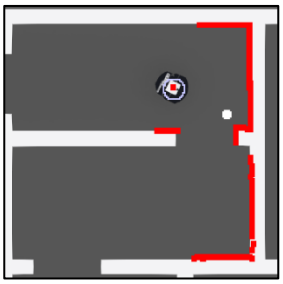
LAB 22 – Laser Range Finders

(1) The goal of this lab is to use a Laser Range Finder to identify doorways and navigate back and forth through a set of open doors in an environment with empty rooms. Download the **Lab22_LaserRangeFinders.zip** file and unzip it. Load up the **Rooms** world which should look as shown below on the left. In Webots, select **Orthographc Projection** from the **View** menu so that the environment looks as shown below on the right.





A **LidarDisplayApp** has been created which will allow you to display the readings from the laser scanner overlayed onto the environment map.



A **LidarProcessor** class is used to represent the laser range finder and its readings. You will want to open the **LidarProcessor** and **Lab22Controller** classes for this assignment, as you will be making changes to the code in there.

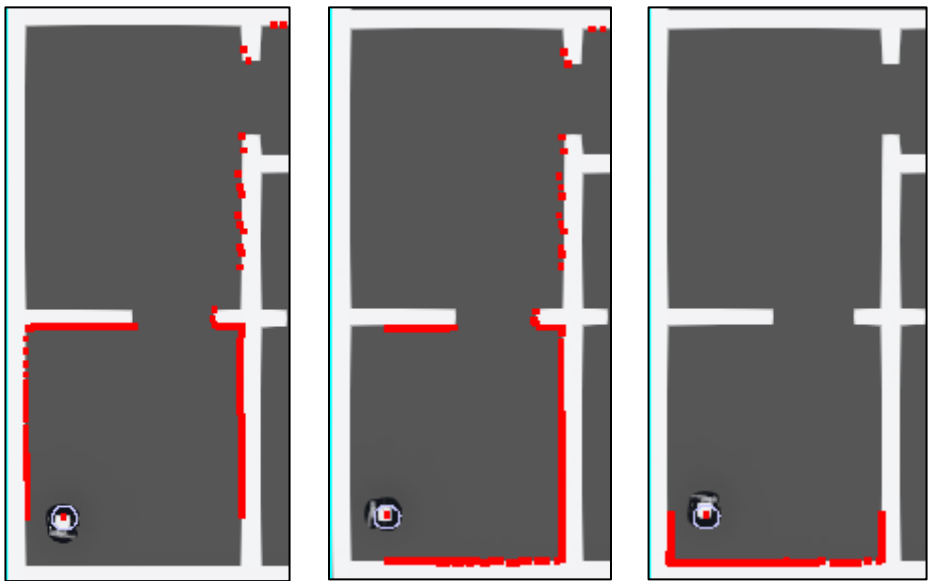
The **Lab22Controller** class has been set up so that the robot can take readings and navigate towards a location of your choosing. There are some methods that you will make use of from this class:

- **getCompassReadingInDegrees()** will return the robot's angle in the environment.
- **turnAway(leftSpeed, rightSpeed)** can be used to turn the robot away from a dangerous collision situation, provided you set the proper motor speeds 😉.
- **moveFrom (x1, y1, x2, y2)** will steer the robot towards point **(x2, y2)**, assuming that it is currently at location **(x1, y1)**.

The code has been set up to provide the actual position of the robot instead of an estimated position. For now, the code simply takes repeated range finder readings and sends them to the **LidarDisplayApp** by calling **applyLidarReading()** with the latest readings and the robot's current angle.

(2) In the **LidarProcessor** class, add code to the **applyReadings()** method so that it applies the given set of 180 lidar readings (passed in as `newRanges` parameter) based on the robot's current angle (passed in as `robotsAngle`). To do this, all you need to do is to fill in the corresponding ranges in the **rangeData** attribute, which is an array that holds range readings for environment angles from 0 to 360 degrees. Since the lidar readings are from 0-179 degrees and are not always facing upwards, you will need to determine where to insert those readings into the **rangeData** array. See slides 9 and 10 for details. Do not remove the call to **resetRangeData()** because it will erase the previous readings each time, so that it will always only show the latest readings. Keep in mind that the incoming range readings come in meters but the **rangeData** array assumes that everything is in centimeters.

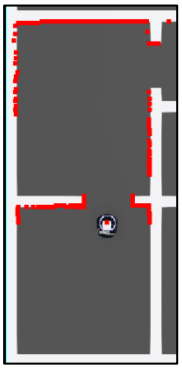
Test your method by running the code. Stretch the display overlay window so that it begins at the top left of the environment and goes to the bottom right. You should see what is shown below on the left. Save a full world image of it (i.e., not just the portion shown) as **Snapshot1.png**. Shrink the display window to get it out of the way, then zoom in on the webots environment a little, click on the robot and then carefully grab the green arrow to spin the robot manually. You might find it easier to go back to the **Perspective Projection** in the **View** menu to manipulate the robot manually like this. If you spin clockwise **90°** (i.e., move mouse up until the simulator indicates **18/12π rad**) you should see the data shown in the middle image below. Spinning an additional **90°** clockwise should give data as shown in the right image below:



Debugging Tips:

- If you get an **ArrayIndexOutOfBoundsException**, then you likely forgot to either…
  - ensure that the index into the **rangeData** array is between **0** and **359** (see caption bubble on slide 10), or …
  - you may have the boundaries wrong in your FOR loop (the range data is an array with indices from **0** to **179**.
- If you DO NOT see any red dots along the wall borders, then you likely forgot to convert from meters to centimeters (see caption bubble on slide 10).

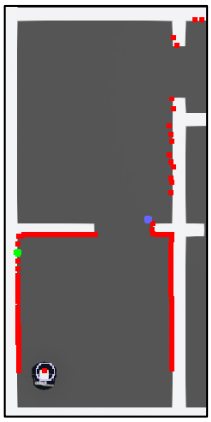
Turn the robot so that it is facing upwards again … you should see the image on the above left again. Now slide the robot sideways by grabbing the tip of the red arrow and sliding it to the right until it is vertically under the doorway above it. Grab the tip of the blue arrow and slide the robot upwards until it is just under the doorway as shown here.



Save a snapshot of the entire environment as **Snapshot2.png**.

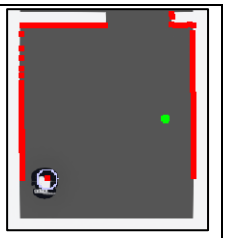
If you want, you can move the robot around to various positions and spin it at various angles to make sure that you understand how the lidar works.

(3) Now we will compute the doorway points. In the **main()** method of the **Lab22Controller**, after the readings are obtained (i.e., where it says WRITE YOUR CODE HERE) … go through the readings to see if you can identify the left and right doorway points (see slide 13). You should use a threshold of **ADJACENCY_TOLERANCE** when checking consecutive readings during your search. If you found BOTH a left point **(leftX, leftY)** AND a right point **(rightX, rightY)** that you consider to be doorway points, then call the method **displayApp.setDoorwayPoints(leftX, leftY, rightX, rightY, x, y)** to allow them to be displayed, where (**x**, **y**) is the robot's location. If you did not find two doorway points, then call **displayApp.setDoorwayPoints(0,0,0,0, x, y)** to make sure that nothing is displayed. You should see what is shown here on the right. This is not actually a "doorway", but it satisfied the adjacency conditions. We will add one more condition to fix this.

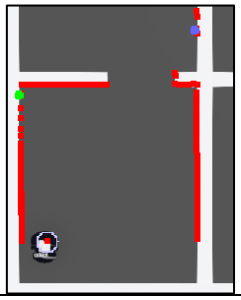


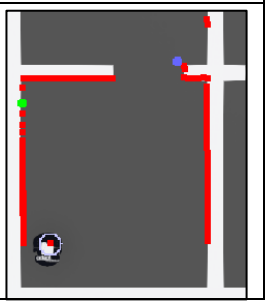
Debugging Tips:

- If you see only one of the dots (or none) … then you have computed something wrong. Likely the dot will also be in the wrong location. A couple of things could be wrong:
  - You may have forgotten to convert to radians before calling sin or cos.
  - You may have forgotten to convert from meters to cm

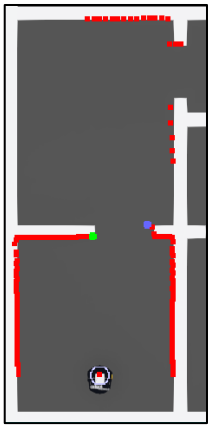


- If you get an **ArrayIndexOutOfBoundsException**, then you are going beyond the array boundaries. Check to ensure that you are not accessing the array at index **180**, since it only goes from **0** to **179**. Also, check to make sure you are not accessing at **-1**. The exception error itself will tell you the value that you are trying to access.
- If you are getting different doorway points as shown here on the left, then you are just off by **1** array index. The right image is to just here to show you the correct points so that you can compare side by side.

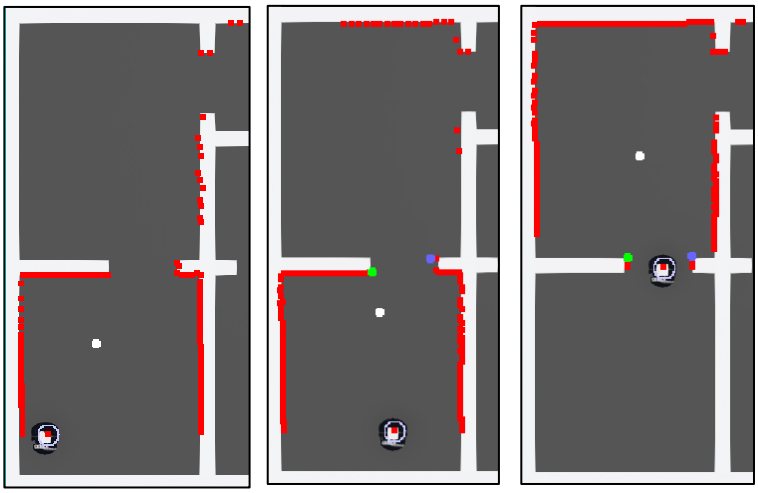




Add code so that the distance between the left (green) and right (blue/purple) doorway points MUST be **≥ 70cm** and **≤ 110cm** to be considered doorway points. Run the code again … you will likely not see the doorway points. Shrink the display window so that you can grab the robot. Click on the robot and slide it manually over to the right a little (red arrow). Then click away from the robot (perhaps onto the background) to deselect it and stretch your display window to overlay perfectly onto the environment again. You should see what is shown here on the right. Take a snapshot of the whole environment with this working doorway as **Snapshot3.png**.



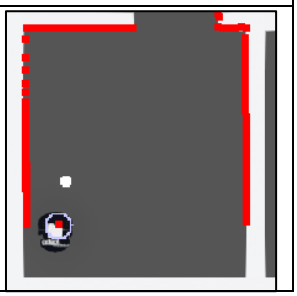
(4) Now we will compute the geometric center point so that we can steer towards it to travel round a little (see slide 15). Add code to the **main()** method again so that it computes the geometric center of the current lidar readings (see slide 16). Call the **displayApp.setWayPoint(centerX, century, x, y)** to have this center displayed relative to the robot's location. Run the code. You should see what is shown below on the left … the white dot represents the geometric center. Save a snapshot (of the whole environment) as **Snapshot4.png**. Shrink the display window so that you can grab the robot. Click on the robot and slide it to the right (red arrow) so that it is centered horizontally in the middle of the door frame. Then click away from the robot (perhaps onto the background) to deselect it and stretch your display window to overlay perfectly onto the environment again. You should see what is shown in the middle below. Notice the white dot has shifted up and to the right a little. Slide the robot upwards so that it is in the middle of the doorway. You should see something like the image below on the right.



Debugging Tips:

- If you do not see any geometric center point appearing (i.e., the white dot), then there could be one of a few things to check:
  - Did you make sure to convert to radians before calling **sin** and **cos**?
  - Did you make sure to convert your readings from **meters** to **cm**?
  - Did you remember to divide by **180** to get the average?

- If you get a center point that looks like what is shown here, then you probably forgot to do the last line of slide 16 to translate the center relative to the robot's location.



(5) Now we will cause the robot to navigate by heading towards the geometric center point. The variables **x** and **y** are given to you as the robot's location. At the end of the **main()** function's **while** loop, you can simply call **moveFrom (x, y, centerX, centerY)** to cause the robot to move from its location to the geometric center location. You should notice that it now successfully navigates through the doorways because of some carefully chosen wheel speed values in the **moveFrom()** method. The robot will likely be able to go all the way through to the top left room and then back again to the bottom left room.

You will notice, however, that the robot crashes into the wall when it gets back to the starting room at the bottom left. Also, it comes quite close to the walls at times when travelling. You should add some additional code that will prevent the robot from crashing head-on into (or rubbing up against) a wall. Do this by checking the lidar reading straight ahead of the robot and the two readings at either side of the robot and then calling the **turnAway()** method when it is "too close", instead of heading towards the geometric center. A **TOO_CLOSE_DISTANCE** and a **TOO_CLOSE_SIDE_DISTANCE** constants have been set that you can use, but you may adjust these if you want to. You just need to choose some left and right motor speeds for the **turnAway()** method call.

To test it, you can always drag and rotate the robot to the place that you want to test it. Alternatively, you can use the fast forward mode so that you don't have to wait too long for the robot to reach the location that you want to test.

Submit **ALL of your code**, including the files that were given to you … as well as the **4** snapshot **.png** files. Make sure that your name and student number is in the first comment line of your code.