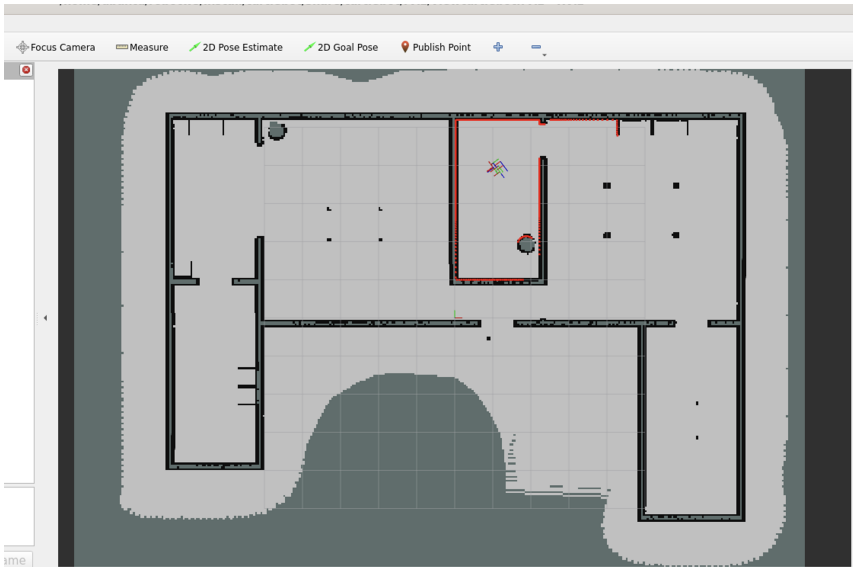
CS 133b **February 13, 2024**

Problem Set 5

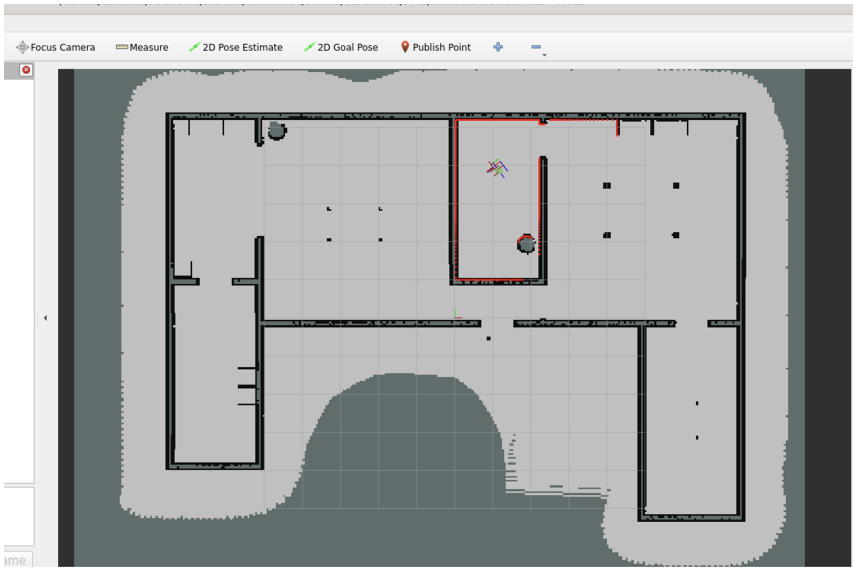
Problem 1 (Welcome to Gazebo and Turtlebot) - 7 points:

Since I had a macbook with an m1 chip, I did not run the code with gazebo. Instead I used the ros2 bag playback command and rviz to visualize. Below is a screenshot of the result. (The launch file is edited as shown in problem 2.)



Problem 2 (Visualize in RVIZ) - 7 points:

Since I had a macbook with an m1 chip, I did not run the code with gazebo. A screenshot is shown below of the output. Screenshots of the edited launch file is shown on the next page.



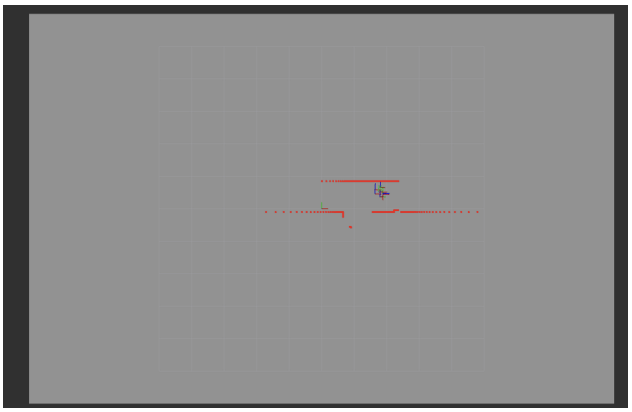
Launch File:

```
def generate_launch_description():

    ########################################################################
    # LOCATE FILES

    # WORLD: Locate the Gazebo world.  Regular house or alternate world.
    #worldfile = os.path.join(pkgdir('turtlebot3_gazebo'),
    #                         'worlds', 'turtlebot3_house.world')
    # worldfile = os.path.join(pkgdir('turtlebot3_gazebo'),
    #                          'worlds', 'turtlebot3_dqn_stage4.world')

    # MODEL: Locate the Gazebo TurtleBot model: With clean or noisy lidar.
    # The original is: turtlebot3_gazebo/models/turtlebot3_waffle/model.sdf
    modelfile = os.path.join(pkgdir('turtlebot'), 'models', 'turtlebot.sdf')
    cleanlidarmodelfile = os.path.join(pkgdir('turtlebot'), 'models',
                                       'turtlebot_cleanlidar.sdf')
    noisylidarmodelfile = os.path.join(pkgdir('turtlebot'), 'models',
                                       'turtlebot_noisylidar.sdf')

    # URDF: Locate and load the TurtleBot URDF, colored orange to stand out.
    # The original is: turtlebot3_description/urdf/turtlebot3_waffle.urdf
    urdffile = os.path.join(pkgdir('turtlebot'), 'urdf', 'turtlebot.urdf')
    with open(urdffile, 'r') as file:
        robot_description = file.read()

    # MAP: Locate the good map
    mapfile = os.path.join(pkgdir('turtlebot'), 'maps/goodmap.yaml')

    # RVIZ: Locate the RVIZ configuration file.
    rvizcfg = os.path.join(pkgdir('turtlebot'), 'rviz/viewturtlebot.rviz')

    # BAGS: Locate the BAG folder.  The first five are in the main house.
    # The alternate recordings use the alternate world (see above).
    #bagfolder = os.path.join(pkgdir('turtlebot'), 'bags', 'singleroom_cleanlaser')
    #bagfolder = os.path.join(pkgdir('turtlebot'), 'bags', 'leftside_cleanlaser')
    bagfolder = os.path.join(pkgdir('turtlebot'), 'bags', 'rightside_cleanlaser')
    #bagfolder = os.path.join(pkgdir('turtlebot'), 'bags', 'singleroom_noisylaser')
    #bagfolder = os.path.join(pkgdir('turtlebot'), 'bags', 'leftside_noisylaser')

    #bagfolder = os.path.join(pkgdir('turtlebot'), 'bags', 'alternate_cleanlaser')
    #bagfolder = os.path.join(pkgdir('turtlebot'), 'bags', 'alternate_noisylaser')
```

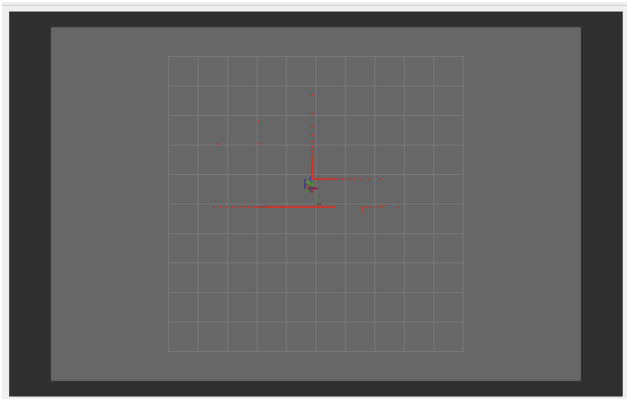
```
# Return the description, built as a python list.
return LaunchDescription([

    # SIMULATION: Start the simulation. If you use Gazebo, you
    # have to start the server and ONE of the spawn nodes (with a
    # clean or noisy lidar).  If you do NOT use Gazebo, start the
    # playback using the BAG folder selected at the top.
    #incl_gzserver,
    #node_spawn_turtlebot_cleanlidar,
    #node_spawn_turtlebot_noisylidar,
    cmd_playback,

    # VIEWER: Select a viewer.  RVIZ or Gazebo client.
    #incl_gzclient,
    node_rviz,

    # FRAMES: For anything outside Gazebo, you will need the robot
    # state publisher (to read the URDF and place the frames on
    # the robot), as well as ONE of the localizations (to locate
    # the robot in the map): perfect or noisy.
    node_robot_state_publisher,
    node_perfectlocalization,
    #node_noisylocalization,

    # MAP: Use either the first two lines (together, to show the
    # perfect map).  OR the last line to start your code...
    node_map_server,
    node_lifecycle,
    #node_buildmap,

])
```

Problem 3 (Running your Code) - 10 points:

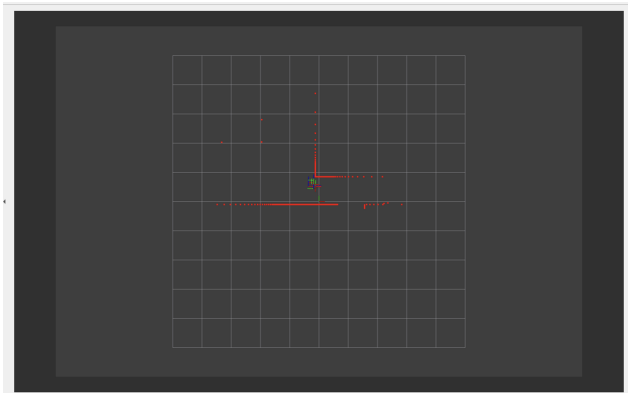
Log odds ratio is initialized to -1 (output shown below):



Log odds ratio is initialized to zero (output shown below):



Log odds ratio is initialized to +1 (output shown below):

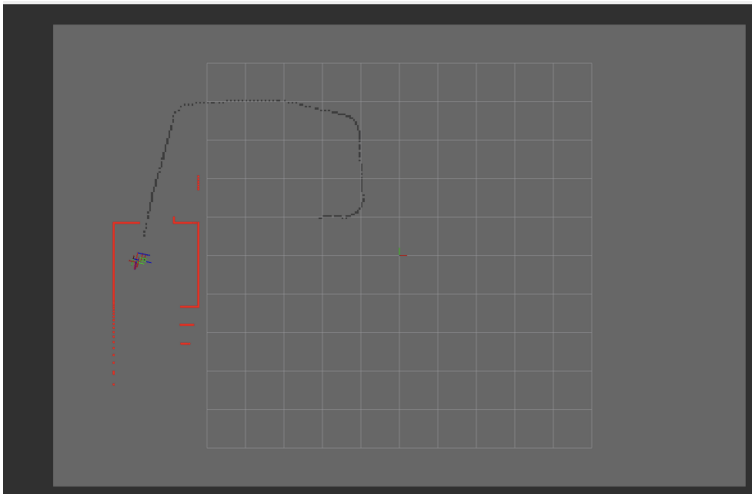


Code for problem 3:

```
def sendMap(self):
    # Convert the log odds ratio into a probability (0...1).
    # Remember: self.logsoddsratio is a 360x240 NumPy array,
    # where the values range from -infinity to +infinity.  The
    # probability should also be a 360x240 NumPy array, but wi
    # values ranging from 0 to 1, being the probability of a w
    #FIXME: Convert the log-odds-ratio into a probability.
    probability = np.zeros((HEIGHT, WIDTH))
    for r in range(HEIGHT):
        for c in range(WIDTH):
            e_l = np.exp(self.logoddsratio[r,c])
            probability[r,c] = e_l / (1 + e_l)
            #print("Prob: ", probability[r,c])
```

Problem 4 (Tracing the Robot) - 14 points:

Below is an example of the output of the trace (using the bag 'leftside_cleanlaser'):

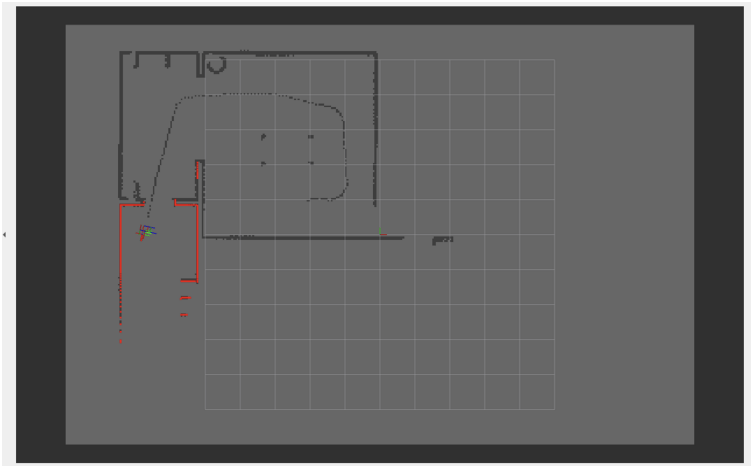


Code added to the laserCB function:

```
################################################################
# FIXME: PROCESS THE LASER SCAN TO UPDATE THE LOG ODDS RATIO!
################################################################
u = int((yc+6.0)/0.05)
v = int((xc+9.0)/0.05)
self.logoddsratio[u,v] = 1.0
```

Problem 5 (Marking the Laser Contact Points) - 17 points:

Below is an example of the output (using the bag 'leftside_cleanlaser'). Note that the trace from problem 4 is still shown below:



Code added to the laserCB function:

```
#############################################################
# FIXME: PROCESS THE LASER SCAN TO UPDATE THE LOG ODDS RATIO!
#############################################################
u = int((yc+6.0)/0.05)
v = int((xc+9.0)/0.05)
self.logoddsratio[u,v] = 1.0

for i in range(len(ranges)):
    r = ranges[i]
    theta = thetas[i]
    if r < rmax and r > rmin:
        alpha = theta + thetac
        dx = r * np.cos(alpha)
        dy = r * np.sin(alpha)
        obj_x = xc + dx
        obj_y = yc + dy
        obj_u = int((obj_y+6.0)/0.05)
        obj_v = int((obj_x+9.0)/0.05)
        self.logoddsratio[obj_u,obj_v] = 1.0
```

Problem 6 (Actual Mapping) - 25 points:

problem 6a
Let l denote the log odds ratio

$$p(m) = 0.90 = 1 - \frac{1}{1+e^l}$$

$$\frac{1}{1+e^l} = 1 - 0.90 = 0.10$$

$$1 = 0.10(1+e^l) \Rightarrow 10 = 1 + e^l \Rightarrow 9 = e^l \Rightarrow ln(9) = l$$

Hence the log odds ratio should be equal to $ln(9) = 2.19722$

problem 6b
Since I did not have gazebo, I approximated the speed of the robot with the following code. (Assuming a frequency of 5hz for the scanner):

```
#self.p was initialized to None
#the code below was used to calculate the speed
dt = 1/5
'''if self.p is None:
    self.p = (xc,yc)

else:
    dx = xc - self.p[0]
    dy = yc - self.p[1]
    dist = np.sqrt(dx**2 + dy**2)
    speed = dist/dt
    print("Speed:{}".format(speed))
    self.p = (xc,yc)'''
```

With the code above, a speed of between 0.2 m/s and 0.3 m/s was found. Therefore to calculate the number of times a ray hits a wall, a speed of 0.25 m/s was used. Using the rmax value in the code, the number of times a way hits a wall (assuming the robot is driving at nominal forward speed, and all scans can be processed) is 70. If we double the robot speed and assume only half of the scans can be processed, the the number of times the ray hits a wall is 17.5.(Code shown below)

```
# speed was between 0.2 m/s and 0.30 m/s
# therefore a speed of 0.25m/s is used
# code below calculates the number of times
# a ray will hit a wall (calculcated to be 70)
# if num_ray_hit_wall is divided by two and speed is doubled
# then num_ray_hit_wall becomes 17.5

'''speed = 0.25 * 2 # in m/s
time_to_hit_wall = rmax/speed #in seconds
num_ray_hit_wall = time_to_hit_wall * 1/dt * 0.50
print("Num {}".format(num_ray_hit_wall))'''
```

Problem 6 (Actual Mapping) - 25 points:

problem 6c
Using the answer from 6b, I decided to use a number between 17.5 and 50. So using 43.575, and the answer from part a, we have:

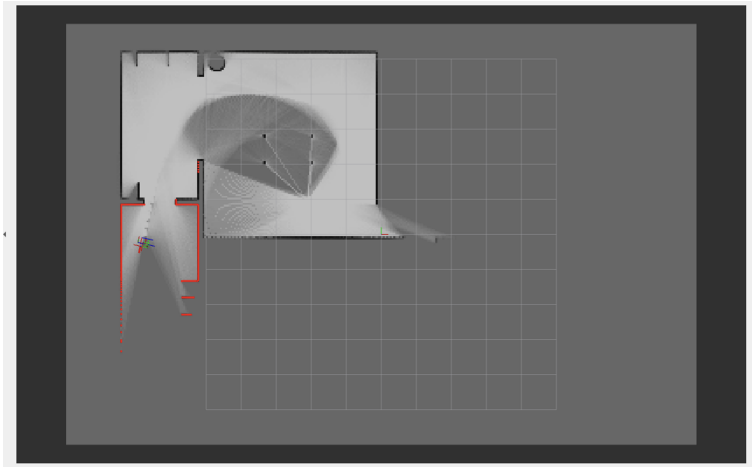
$$l_{occupied} = \frac{ln(9)}{50} = \frac{2.19722}{43.575} = 0.044$$

problem 6d
As stated in the problem, because rays are much more tightly spaced near the robot and hence the grid elements near the robot are seen by many more rays, the magnitude of l_{free} should be less than the magnitude of $l_{occupied}$. In other words, this is equivalent to assuming more scans can be processed so we have:

$$l_{free} = \frac{ln(9)}{70} = \frac{2.19722}{70} = 0.0313$$

Problem 6 (Actual Mapping) - 25 points:

Below is an example of the output (using the bag 'leftside_cleanlaser').

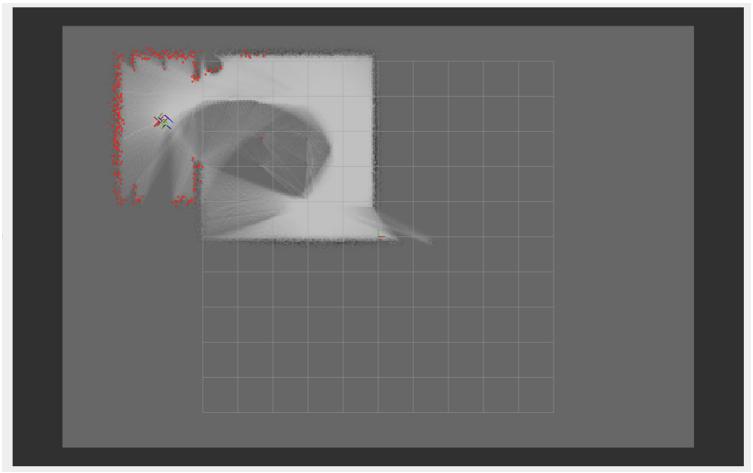


Code in the laserCB function:

```
############################################################
# FIXME: PROCESS THE LASER SCAN TO UPDATE THE LOG ODDS RATIO!
############################################################
u = int((yc+6.0)/0.05)
v = int((xc+9.0)/0.05)
#self.logoddsratio[u,v] = 1.0

for i in range(len(ranges)):
    r = ranges[i]
    theta = thetas[i]
    if r < rmax and r > rmin:
        alpha = theta + thetac
        dx = r * np.cos(alpha)
        dy = r * np.sin(alpha)
        obj_x = xc + dx
        obj_y = yc + dy
        obj_u = int((obj_y+6.0)/0.05)
        obj_v = int((obj_x+9.0)/0.05)
        self.logoddsratio[obj_u,obj_v] += LOCCUPIED

        start = (u,v)
        obj_u = (obj_y+6.0)/0.05
        obj_v = (obj_x+9.0)/0.05
        end = (obj_u, obj_v)
        intermediate_locs = self.bresenham(start, end)
        for (r,c) in intermediate_locs:
            self.logoddsratio[r,c] -= LFREE
```

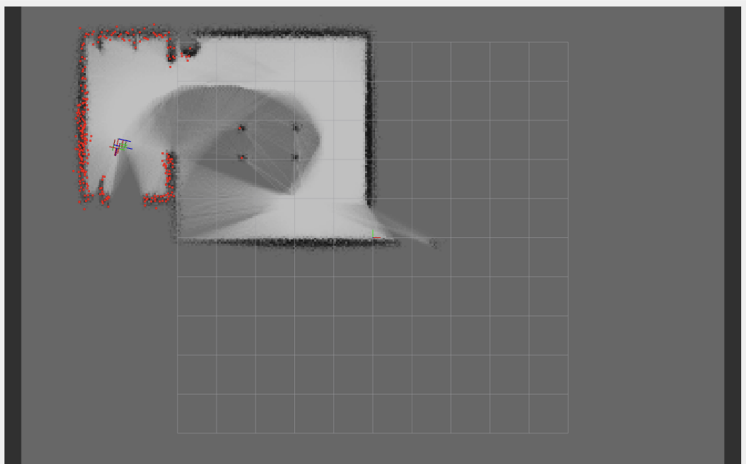
Problem 7 (Testing with Noisy Sensors) - 8 points:

problem 7a
Below is an example of the output (using the bag 'leftside_noisylaser'). Code is the same as in problem 6.

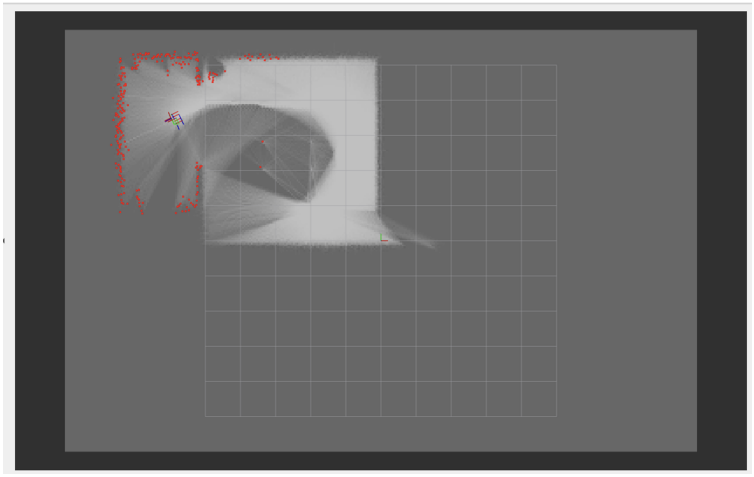


As shown above we can see the points of contact (where rays hit a wall) have a greater variability than before (refer to the red dots in the figure above). This is expected as now there is noise in the lidar sensor. As a result, there is more uncertainty in regards to where the walls are.

problem 7b
Below is an example of the output (using the bag 'leftside_noisylaser'). Code is the same as in problem 6, except $l_{occupied}$ is increased by a factor of 3. As shown below, we now see that the wall is darker, in other words it is easier for a cell to change from free to occupied

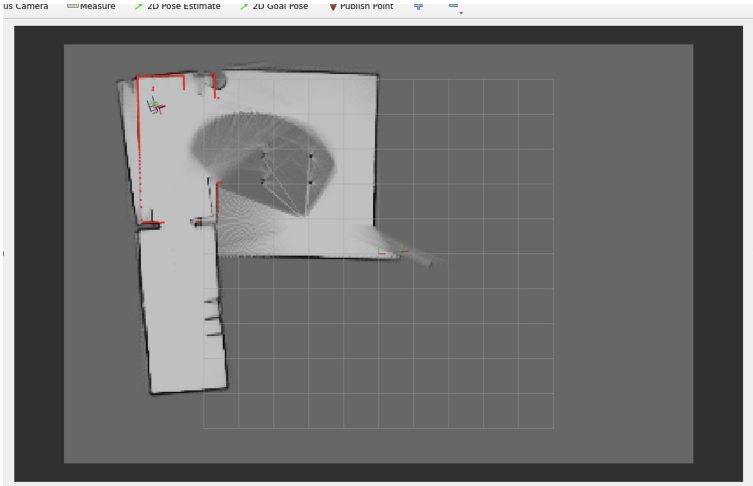


On the next page is an example of the output (using the bag 'leftside_noisylaser'). Code is the same as in problem 6, except $l_{occupied}$ is decreased by a factor of 3. As shown below, we now see that the wall is lighter, in other words it is harder for a cell to change from free to occupied.



Problem 8 (Testing with Bad Localization) - 8 points:

problem 8a Below is an example of the output (using the bag 'leftside_clanlaser') with noisy localization (the launch file was edited accordingly, shown below). The same code was used as in problem 6. As shown below, we see that after the robot moves into a new room and returns to the old room, the walls that the robot detect are shifted/rotated. In the figure this would be comparing the red dots to the darker lines.




```
# Return the description, built as a python list.
return LaunchDescription([

    # SIMULATION: Start the simulation. If you use Gazebo, you
    # have to start the server and ONE of the spawn nodes (with a
    # clean or noisy lidar).  If you do NOT use Gazebo, start the
    # playback using the BAG folder selected at the top.
    #incl_gzserver,
    #node_spawn_turtlebot_cleanlidar,
    #node_spawn_turtlebot_noisylidar,
    cmd_playback,

    # VIEWER: Select a viewer.  RVIZ or Gazebo client.
    #incl_gzclient,
    node_rviz,

    # FRAMES: For anything outside Gazebo, you will need the robot
    # state publisher (to read the URDF and place the frames on
    # the robot), as well as ONE of the localizations (to locate
    # the robot in the map): perfect or noisy.
    node_robot_state_publisher,
    #node_perfectlocalization,
    node_noisylocalization,

    # MAP: Use either the first two lines (together, to show the
    # perfect map).  OR the last line to start your code...
    #node_map_server,
    #node_lifecycle,
    node_buildmap,

])
```

problem 8b One way to fix this can be to detect when a loop has been made (when the robot has returned to an area that is has visited before). Then we can compare the walls the robot detects with the old walls. If the geometry is similar, we can determine or have a probability that favors the walls being unchanged. In other words, we are essentially doing some sort of recognition of the places that have been visited.

Problem 10 (Time Spent) - 4 points:

I spent about 4.5 hours on the problem set. It should be noted that I have macbook with an m1 chip and therefore did not install gazebo.