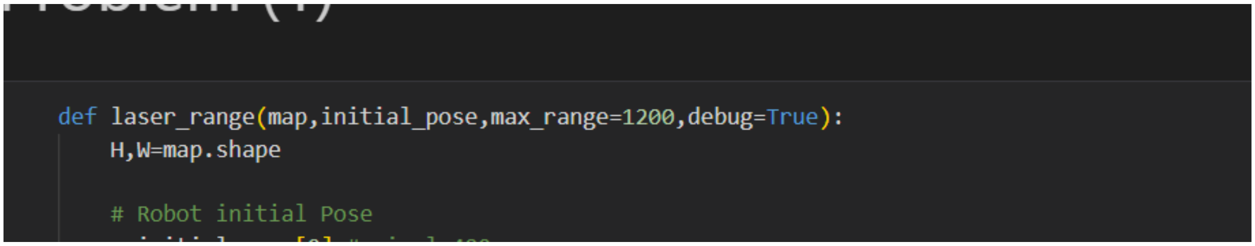
Assignment (1)

Name	Basma Hatem Elhoseny
Sec	1
BN	16
Code	9202381

Presented to: Eng Mohamed Shawk

Requirement(1):

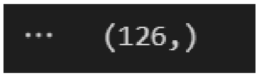


Problem (1)

```
def laser_range(map,initial_pose,max_range=1200,debug=True):
    H,W=map.shape

    # Robot initial Pose
```

Given the initial pose of the robot I want to take the readings if the lidar with max_range as passed

Return of this function is the reading vector of dimensionality corresponding to each angle of rotation by the sensor

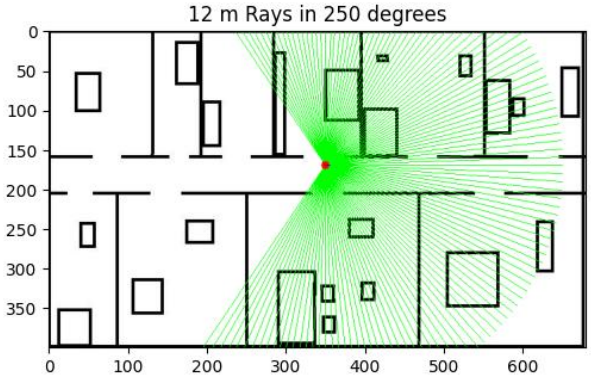


```
...   (126,)
```

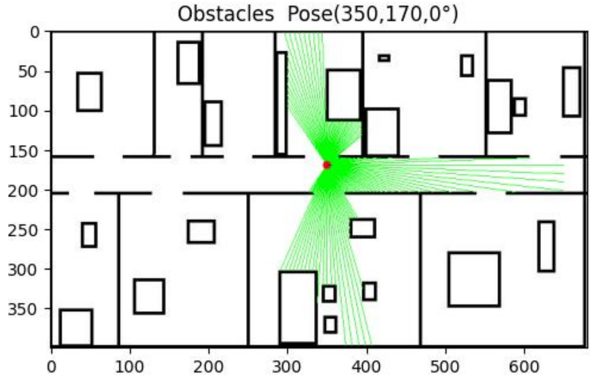
Output:

The left images will be the max_range of the rays without taking inconsideration the obstacles and the right images are the actual readings taken by the sensor from the nearest obstacle :D

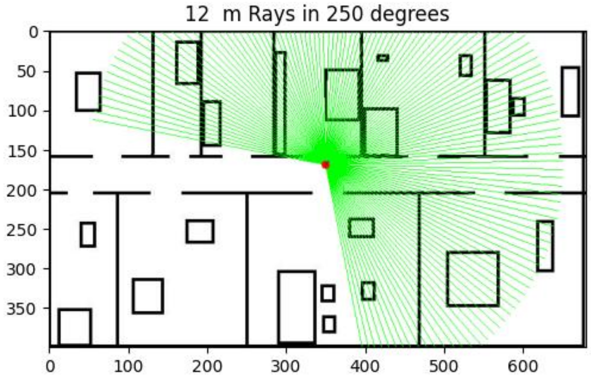
12 m Rays in 250 degrees



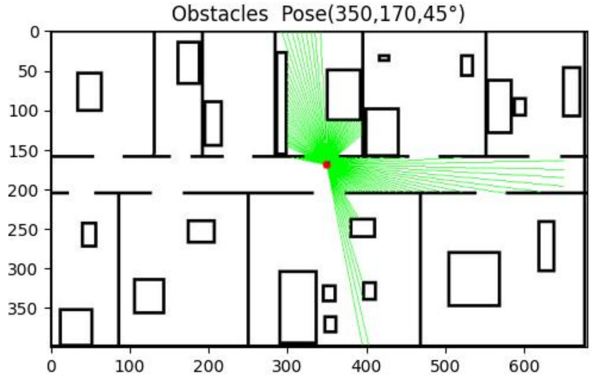
Obstacles Pose(350,170,0°)



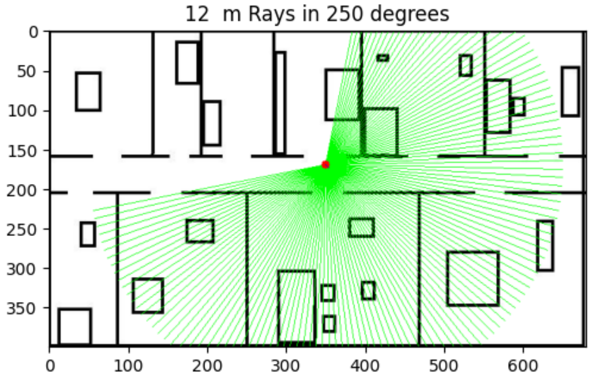
12 m Rays in 250 degrees



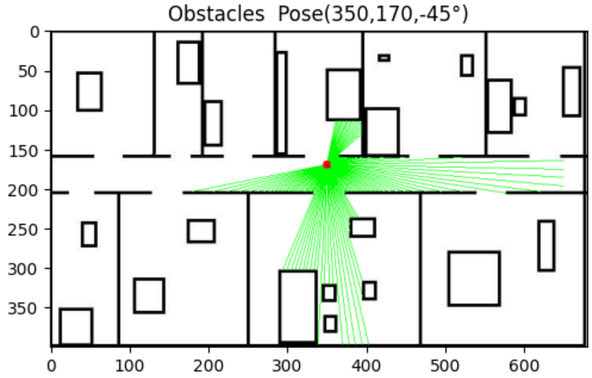
Obstacles Pose(350,170,45°)



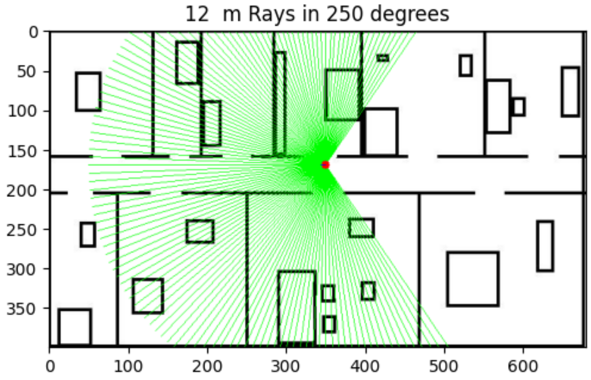
12 m Rays in 250 degrees



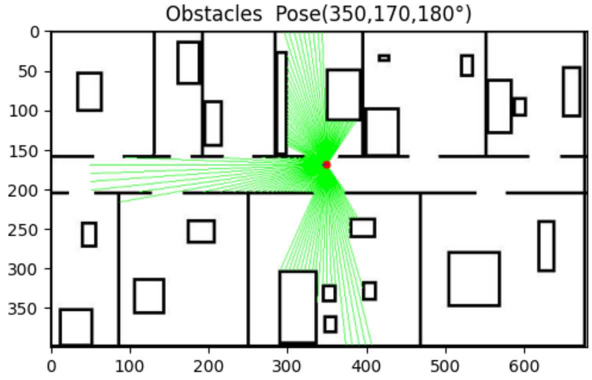
Obstacles Pose(350,170,-45°)

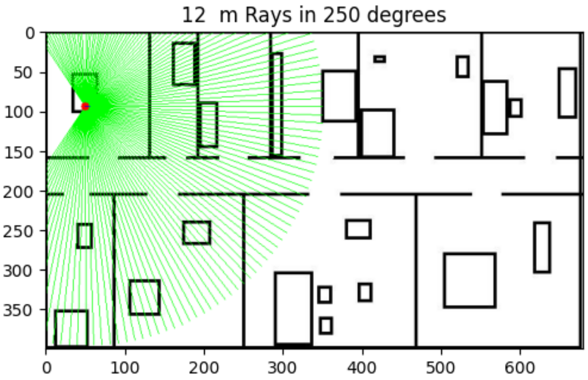


12 m Rays in 250 degrees

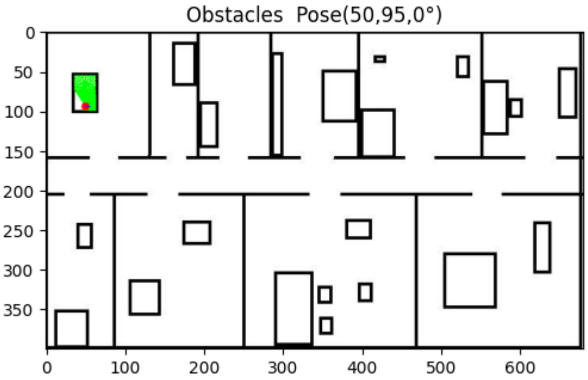


Obstacles Pose(350,170,180°)

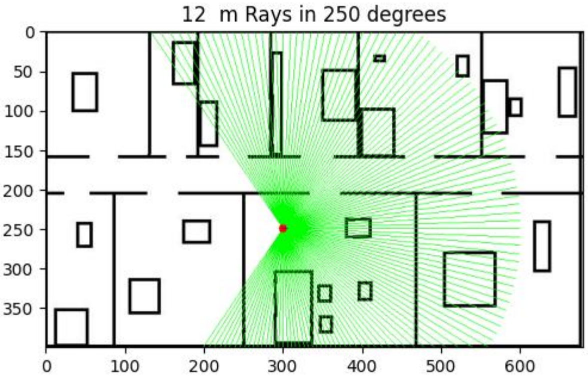




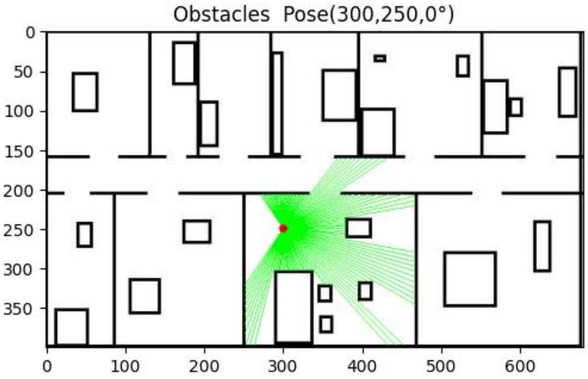
12 m Rays in 250 degrees



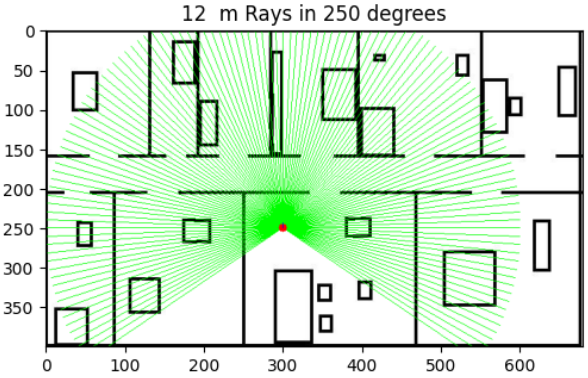
Obstacles Pose(50,95,0°)



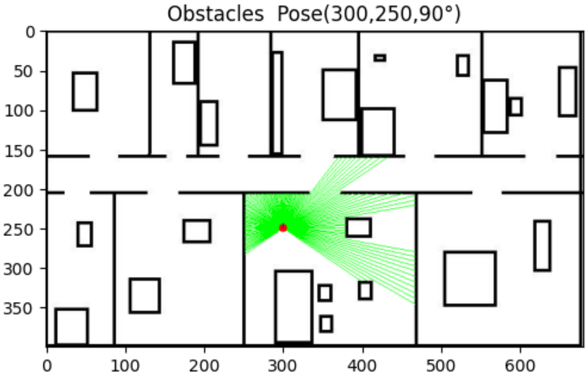
12 m Rays in 250 degrees



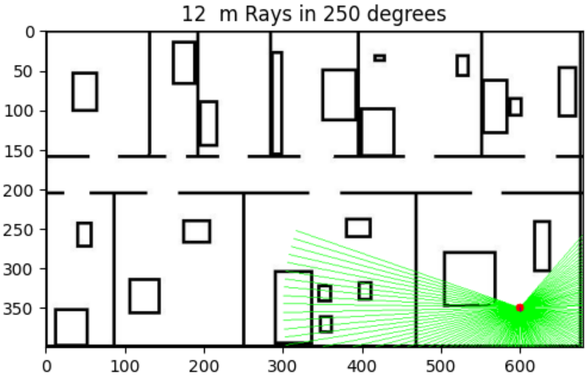
Obstacles Pose(300,250,0°)



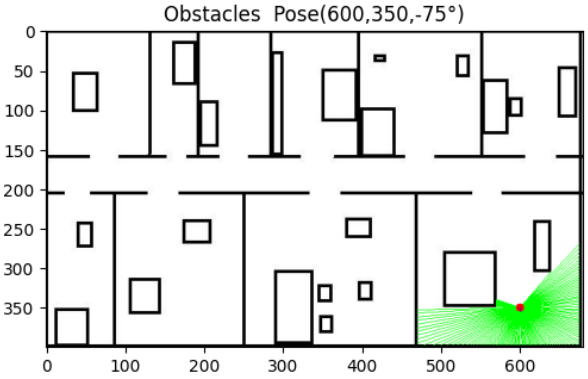
12 m Rays in 250 degrees



Obstacles Pose(300,250,90°)

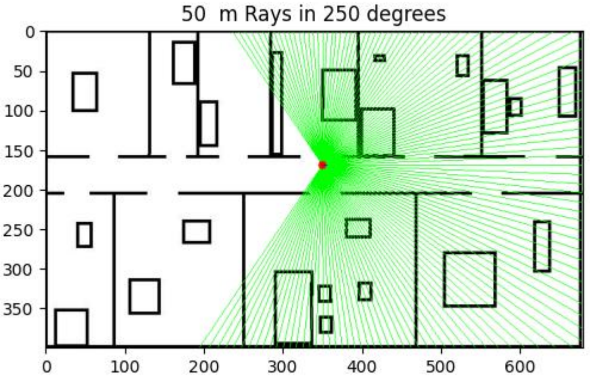


12 m Rays in 250 degrees

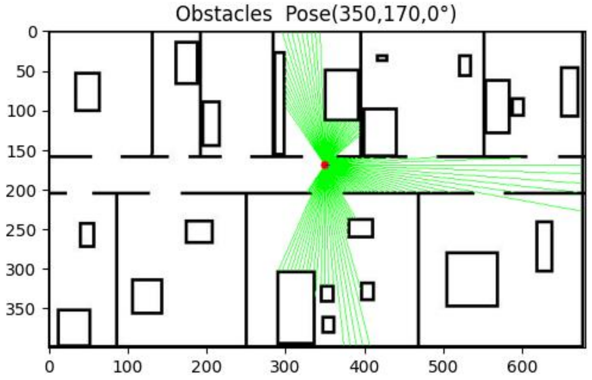


Obstacles Pose(600,350,-75°)

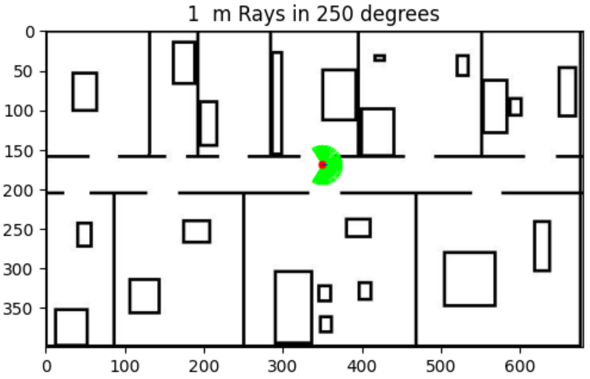
50 m Rays in 250 degrees



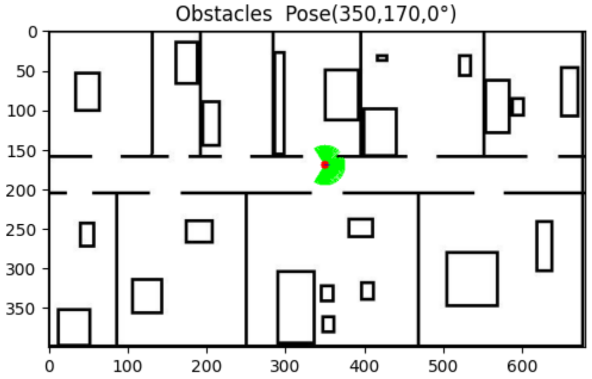
Obstacles Pose(350,170,0°)



1 m Rays in 250 degrees

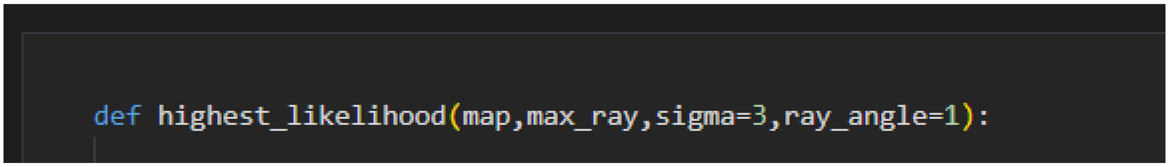


Obstacles Pose(350,170,0°)



Requirement(2):

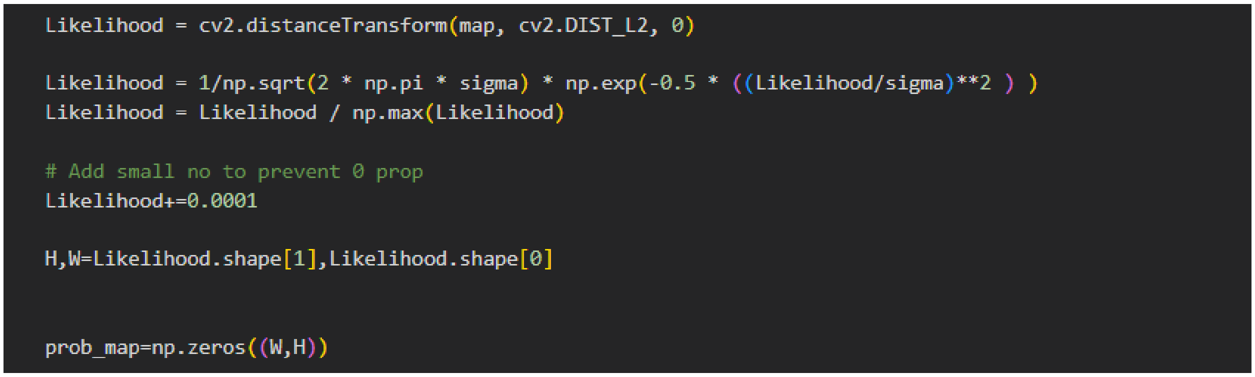
Given certain Sigma and max_ray measurement we need to get the likelihood map



```
def highest_likelihood(map,max_ray,sigma=3,ray_angle=1):
```

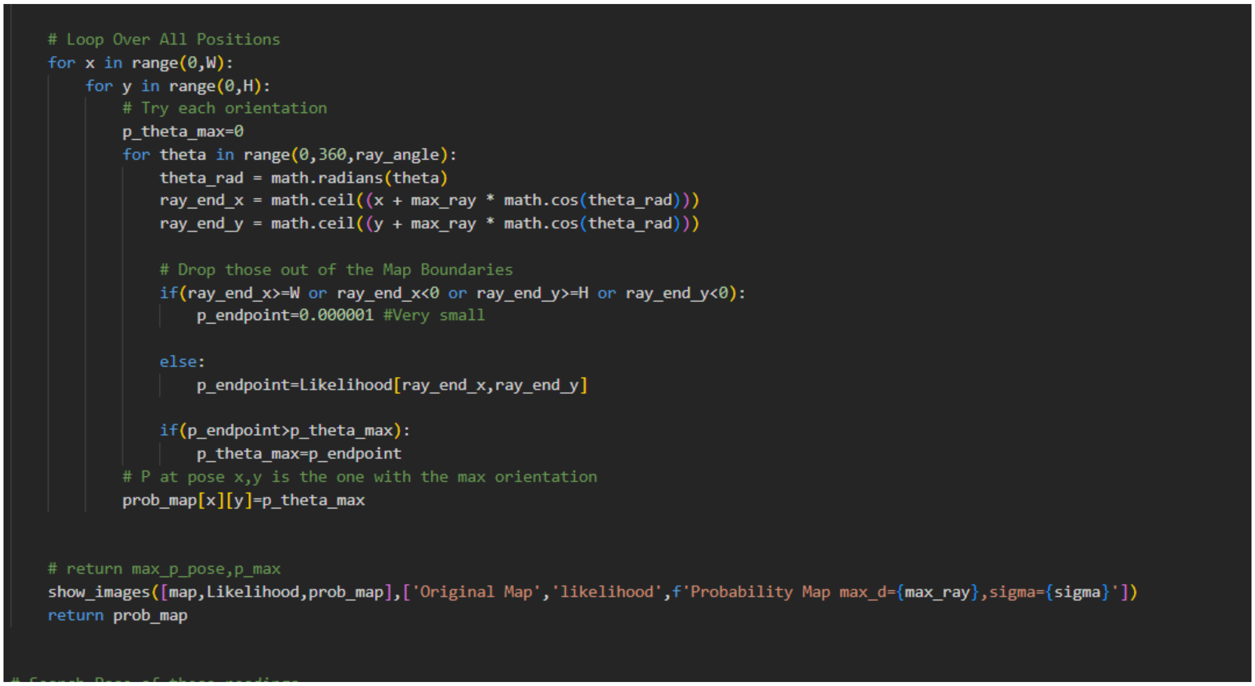
First we need to compute the likelihood

Note: we can still use Gaussian blur here which is a special case of the distance transform especially we are working with images



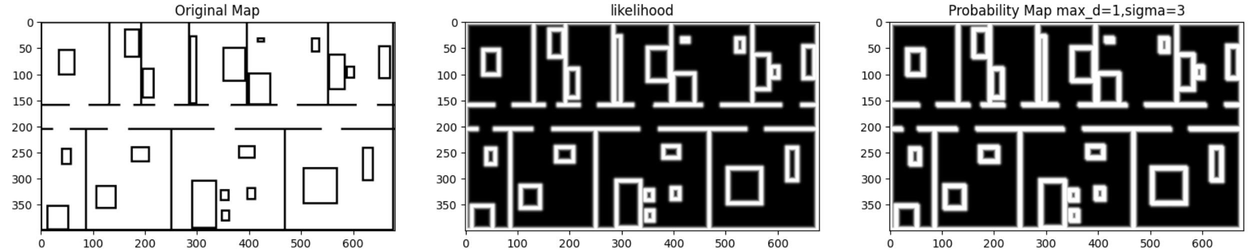
```
Likelihood = cv2.distanceTransform(map, cv2.DIST_L2, 0)

Likelihood = 1/np.sqrt(2 * np.pi * sigma) * np.exp(-0.5 * ((Likelihood/sigma)**2 ) )
Likelihood = Likelihood / np.max(Likelihood)

# Add small no to prevent 0 prop
Likelihood+=0.0001

H,W=Likelihood.shape[1],Likelihood.shape[0]


prob_map=np.zeros((W,H))
```

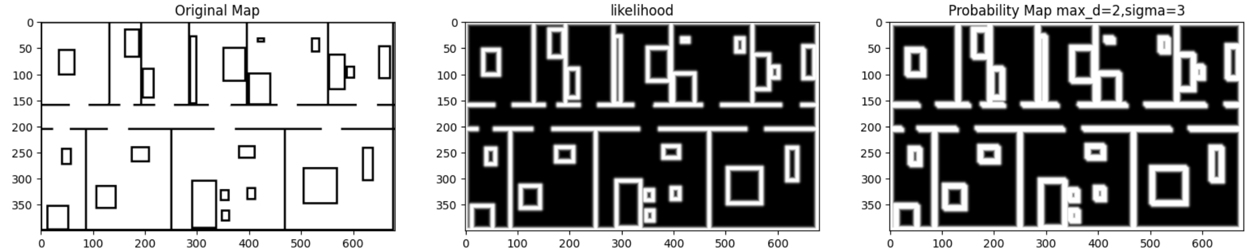
Then we will loop over all the poses and get the most probable orientation given the likelihood

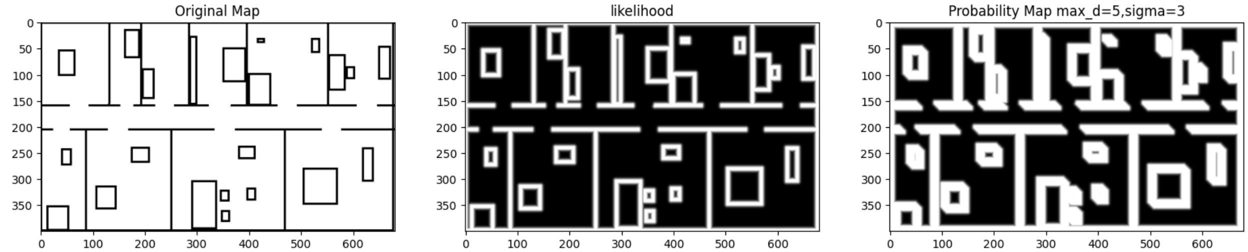


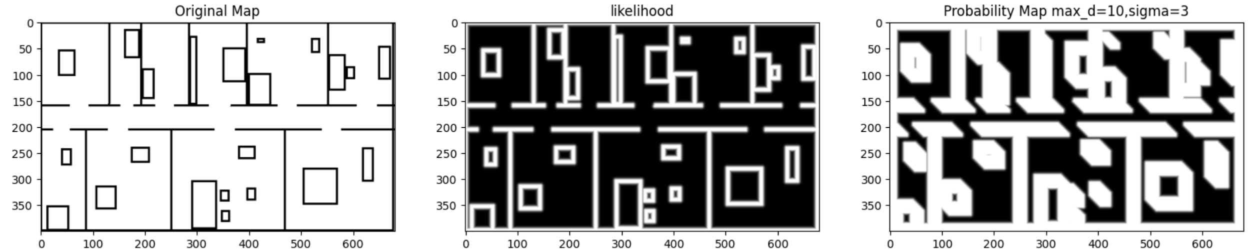
```
# Loop Over All Positions
for x in range(0,W):
    for y in range(0,H):
        # Try each orientation
        p_theta_max=0
        for theta in range(0,360,ray_angle):
            theta_rad = math.radians(theta)
            ray_end_x = math.ceil((x + max_ray * math.cos(theta_rad)))
            ray_end_y = math.ceil((y + max_ray * math.cos(theta_rad)))

            # Drop those out of the Map Boundaries
            if(ray_end_x>=W or ray_end_x<0 or ray_end_y>=H or ray_end_y<0):
                p_endpoint=0.000001 #Very small

            else:
                p_endpoint=Likelihood[ray_end_x,ray_end_y]

            if(p_endpoint>p_theta_max):
                p_theta_max=p_endpoint
        # P at pose x,y is the one with the max orientation
        prob_map[x][y]=p_theta_max


# return max_p_pose,p_max
show_images([map,Likelihood,prob_map],['Original Map','likelihood',f'Probability Map max_d={max_ray},sigma={sigma}'])
return prob_map
```

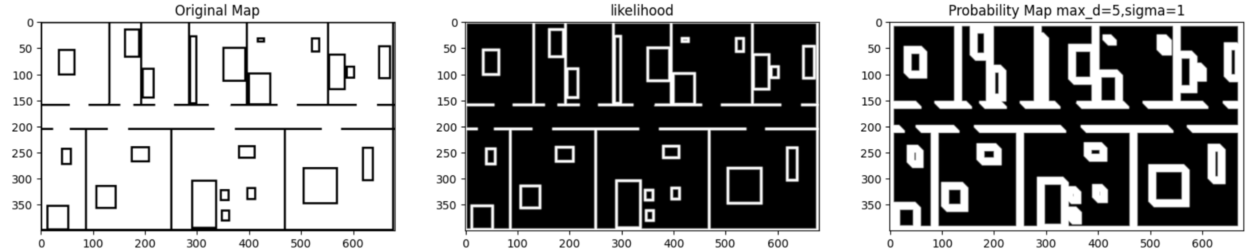
Output:

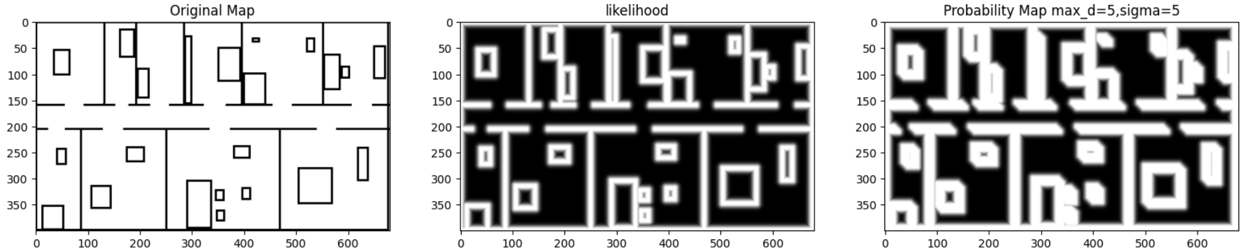

Original Map
likelihood
Probability Map max_d=1,sigma=3
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600

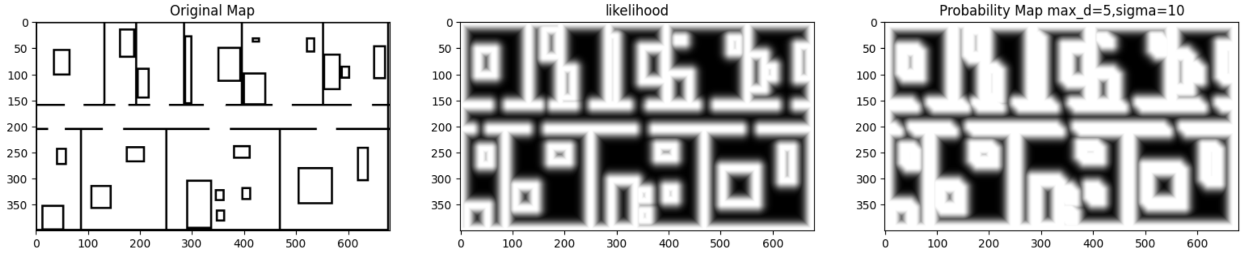

Original Map
likelihood
Probability Map max_d=2,sigma=3
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600


Original Map
likelihood
Probability Map max_d=5,sigma=3
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600


Original Map
likelihood
Probability Map max_d=10,sigma=3
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600


Original Map
likelihood
Probability Map max_d=5,sigma=1
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600

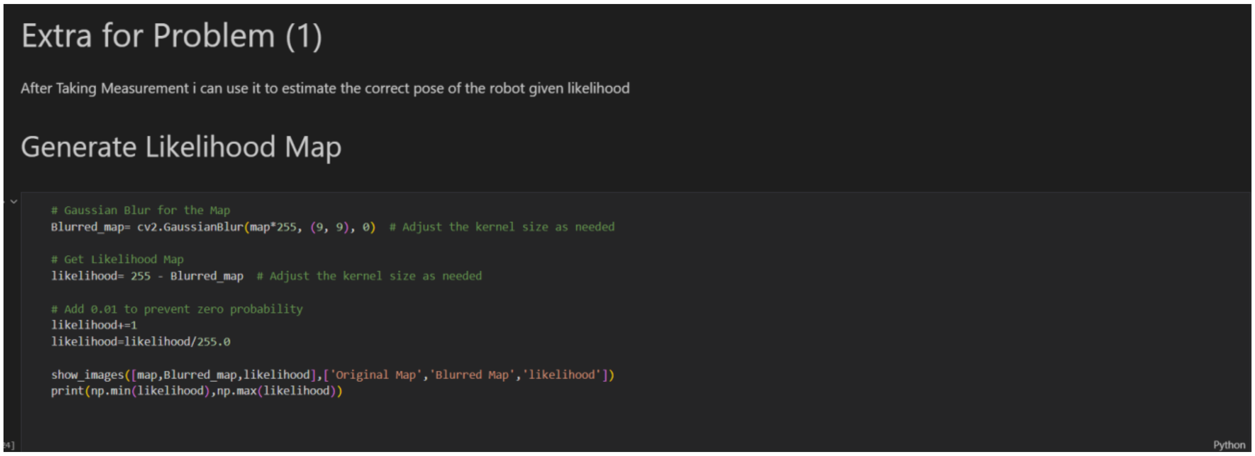

Original Map
likelihood
Probability Map max_d=5,sigma=5
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600


Original Map
likelihood
Probability Map max_d=5,sigma=10
0
50
100
150
200
250
300
350
0
100
200
300
400
500
600

Extra part to be use readings from req(1) to estimate the pose [Not Required]

We generate the likelihood of the map

Then we will take reading of the sensor with that likelihood and estimate the position of the robot in the map [ie keep searching in all the grid at which position can I get these readings 😊😊]

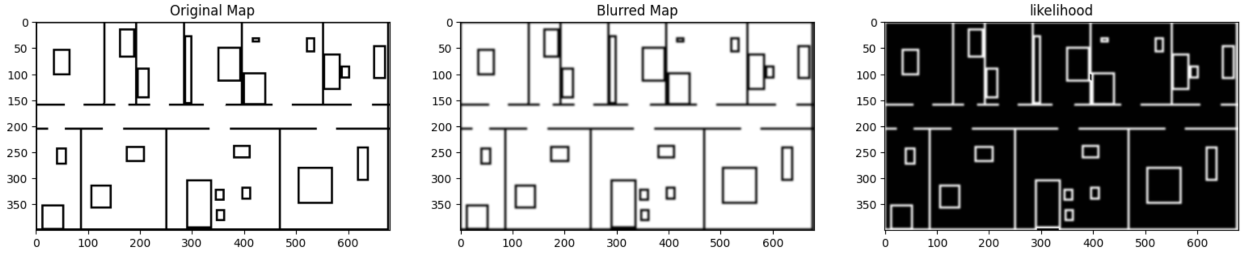


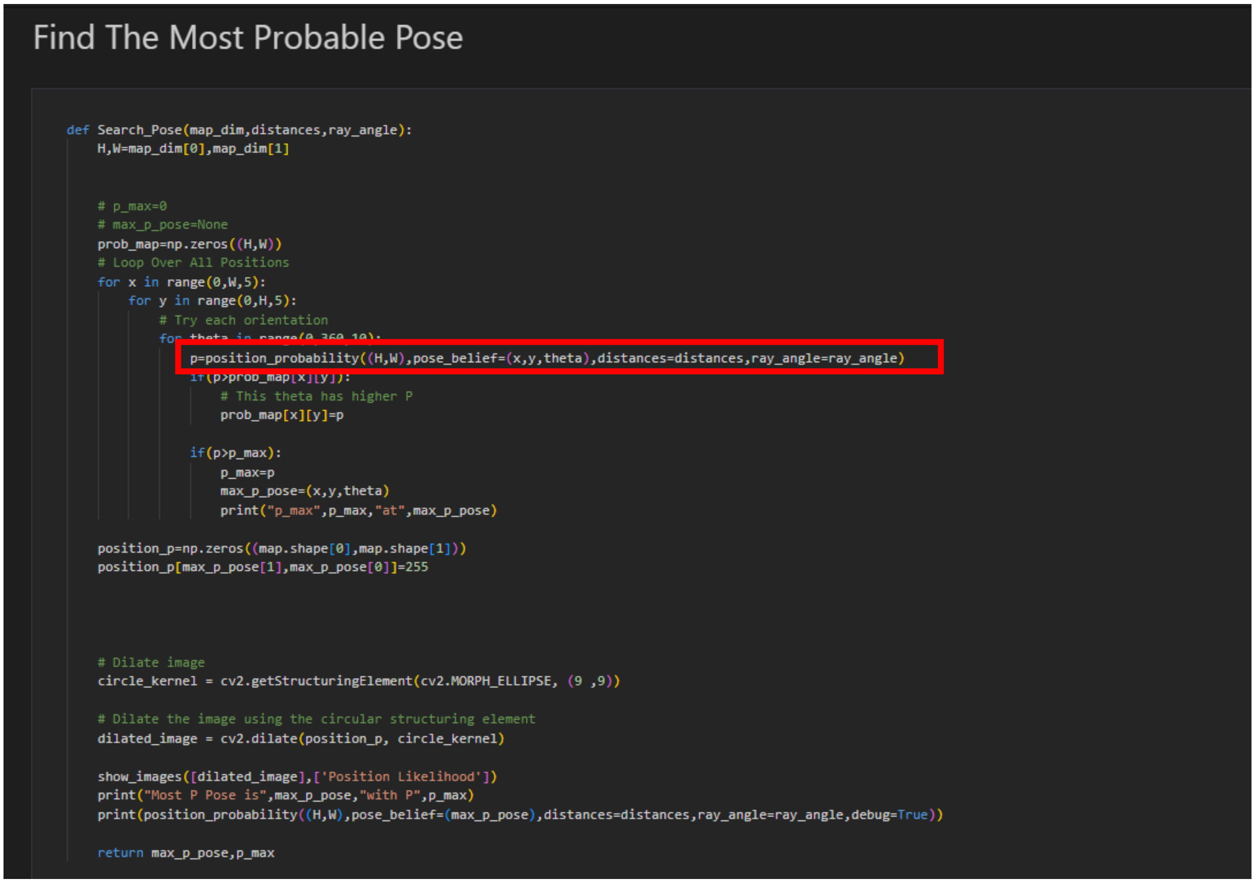
Extra for Problem (1)

After Taking Measurement i can use it to estimate the correct pose of the robot given likelihood

Generate Likelihood Map

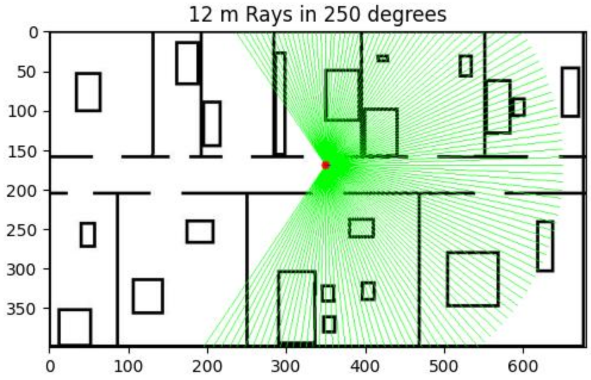
```python
# Gaussian Blur for the Map
Blurred_map= cv2.GaussianBlur(map*255, (9, 9), 0)  # Adjust the kernel size as needed

# Get Likelihood Map
likelihood= 255 - Blurred_map  # Adjust the kernel size as needed

# Add 0.01 to prevent zero probability
likelihood+=1
likelihood=likelihood/255.0

show_images([map,Blurred_map,likelihood],['Original Map','Blurred Map','likelihood'])
print(np.min(likelihood),np.max(likelihood))
```

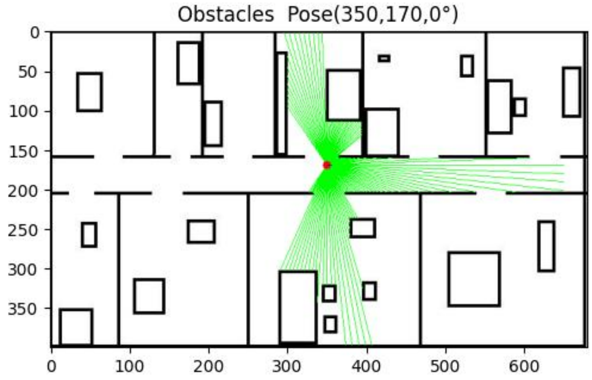




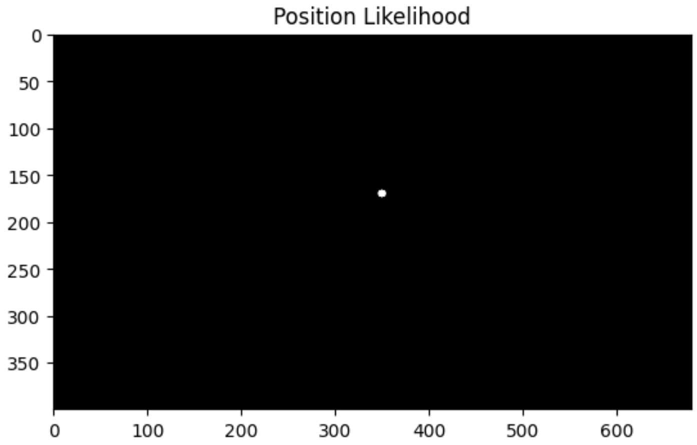
Find The Most Probable Pose

```
def Search_Pose(map_dim,distances,ray_angle):
    H,W=map_dim[0],map_dim[1]


    # p_max=0
    # max_p_pose=None
    prob_map=np.zeros((H,W))
    # Loop Over All Positions
    for x in range(0,W,5):
        for y in range(0,H,5):
            # Try each orientation
            for theta in range(0,360,10):
                p=position_probability((H,W),pose_belief=(x,y,theta),distances=distances,ray_angle=ray_angle)
                if(p>prob_map[x][y]):
                    # This theta has higher P
                    prob_map[x][y]=p

                if(p>p_max):
                    p_max=p
                    max_p_pose=(x,y,theta)
                    print("p_max",p_max,"at",max_p_pose)

    position_p=np.zeros((map.shape[0],map.shape[1]))
    position_p[max_p_pose[1],max_p_pose[0]]=255




    # Dilate image
    circle_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9 ,9))

    # Dilate the image using the circular structuring element
    dilated_image = cv2.dilate(position_p, circle_kernel)

    show_images([dilated_image],['Position Likelihood'])
    print("Most P Pose is",max_p_pose,"with P",p_max)
    print(position_probability((H,W),pose_belief=(max_p_pose),distances=distances,ray_angle=ray_angle,debug=True))

    return max_p_pose,p_max
```



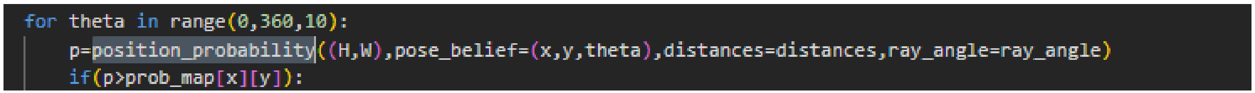


Given that reading we get that the most probable position is

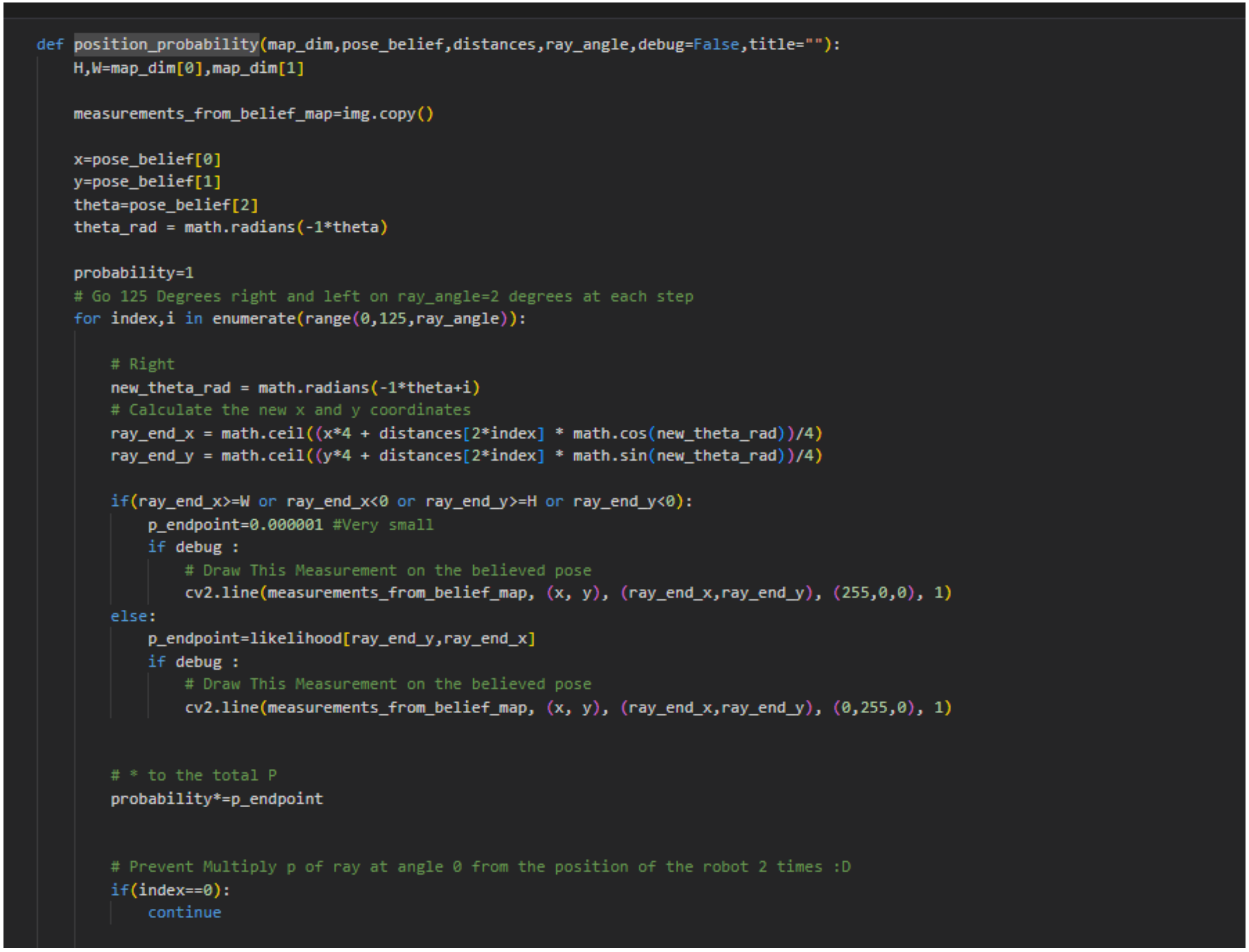


Regarding the function

I computes the P of pose_belief to be the postion the robot is in now given its readings and the likehood map



```
for theta in range(0,360,10):
    p=position_probability((H,W),pose_belief=(x,y,theta),distances=distances,ray_angle=ray_angle)
    if(p>prob_map[x][y]):
```

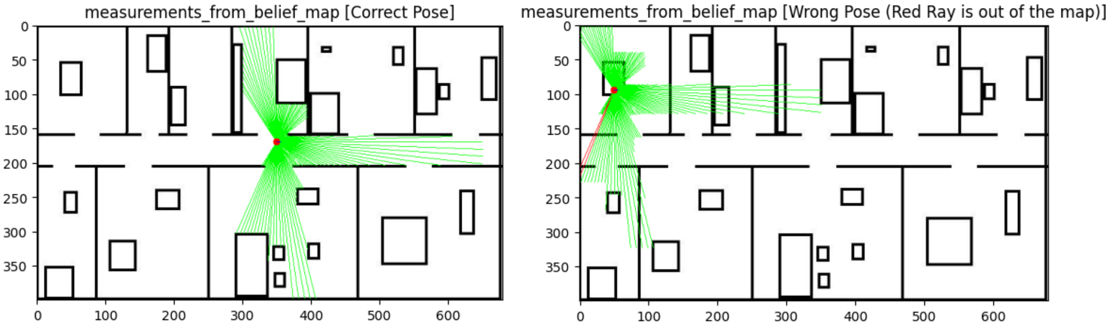


```
def position_probability(map_dim,pose_belief,distances,ray_angle,debug=False,title=""):
    H,W=map_dim[0],map_dim[1]

    measurements_from_belief_map=img.copy()

    x=pose_belief[0]
    y=pose_belief[1]
    theta=pose_belief[2]
    theta_rad = math.radians(-1*theta)

    probability=1
    # Go 125 Degrees right and left on ray_angle=2 degrees at each step
    for index,i in enumerate(range(0,125,ray_angle)):

        # Right
        new_theta_rad = math.radians(-1*theta+i)
        # Calculate the new x and y coordinates
        ray_end_x = math.ceil((x*4 + distances[2*index] * math.cos(new_theta_rad))/4)
        ray_end_y = math.ceil((y*4 + distances[2*index] * math.sin(new_theta_rad))/4)

        if(ray_end_x>=W or ray_end_x<0 or ray_end_y>=H or ray_end_y<0):
            p_endpoint=0.000001 #Very small
            if debug :
                # Draw This Measurement on the believed pose
                cv2.line(measurements_from_belief_map, (x, y), (ray_end_x,ray_end_y), (255,0,0), 1)
        else:
            p_endpoint=likelihood[ray_end_y,ray_end_x]
            if debug :
                # Draw This Measurement on the believed pose
                cv2.line(measurements_from_belief_map, (x, y), (ray_end_x,ray_end_y), (0,255,0), 1)


        # * to the total P
        probability*=p_endpoint


        # Prevent Multiply p of ray at angle 0 from the position of the robot 2 times :D
        if(index==0):
            continue
```

Output:

The left is the visualization of the givien measuremnat and the pose_befilef it is clear that it is correct belief about the postion

The right the pose is wring s that readings doesn’t match with the surronding obstacles so the left has belief higher proboanlity than the one ta the right


measurements_from_belief_map [Correct Pose]
measurements_from_belief_map [Wrong Pose (Red Ray is out of the map)]