Robotics – HW2

Zahra Hashemi – 97243072

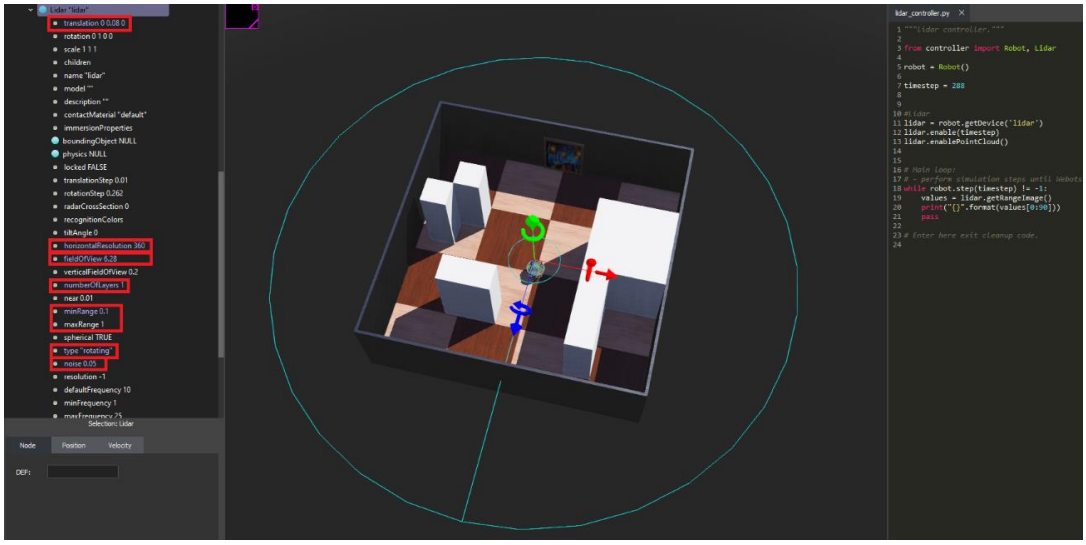
Dorreen Rostami – 97243034

Niloufar Moradi Jam – 97243063

1)

a. First, we collect the data with the lidar. The lidar's properties are set according to the question requirements.

The default timestep in webots is 32 but by this timestep, not all of the lidar's pointClouds are turned on. With trial and error we figured out it should run a timestep 9 times, so we set the timestep inside the code equal to 32*9=288. Thus, it can scan all around properly and then we print it and use the printed data in our Matlab program.

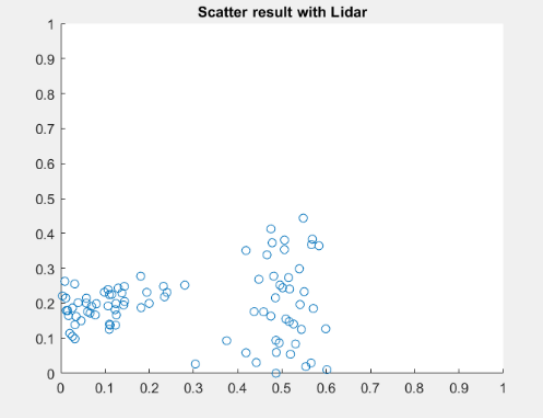


The data consists of the distance of the robot from different points on the wall. In order to find x and y for each point we must use this formula:

$x = r*\cos(\theta)$ $y = r*\sin(\theta)$

```
C q1_1.m
home > nanami > Downloads > C q1_1.m
1   clear;clc;
2   r = importdata('lidar.txt', ',');
3   X = [];
4   Y = [];
5   % x = r cos  y = r sin
6   for i = 1:90
7      x1 = r(i)*cosd(i-1);
8      X = [X, x1];
9      y1 = r(i)*sind(i-1);
10     Y = [Y, y1];
11  end
12
13  %alef
14  %scattering two walls (1:90)
15  figure
16  scatter(X,Y);
17  title('Scatter result with Lidar')
18  axis([0 1 0 1]);
19
```

Then we scatter the result of the above formulas. The angle range is between 0 and 90 in such a way that each value in the array of distances has an angle the size of one degree larger than the one before it:



The result indicates that there are two walls from 0 to 90. The right wall is in the array indices of 0 to 42.

b. The pseudo inverse algorithm according to the slides:

- برای بدست آوردن خط با کمترین خطا از روش شبه‌معکوس (Pseudo Inverse) استفاده می‌کنیم

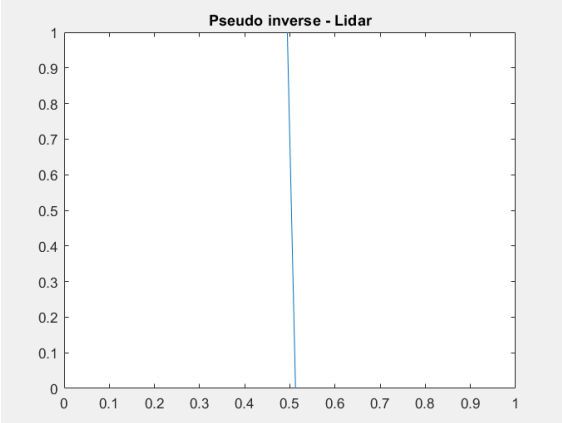
$$Xp = c \Rightarrow X^T Xp = X^T c$$
$$\Rightarrow (X^T X)^{-1} X^T Xp = (X^T X)^{-1} X^T c$$
$$\Rightarrow p = (X^T X)^{-1} X^T c$$

pseudo_inverse.m

```
home > nanami > Downloads > pseudo_inverse.m
function [p] = pseudo_inverse(X, c)
    p = inv(transpose(X)*X)*transpose(X)*c;
end
```

```
%be
% getting just one wall
X = X(1:42);
Y = Y(1:42);
Z = [X.',Y.'];
c = ones(size(X.'));
p = pseudo_inverse(Z, c);
disp(p);
writematrix(Z,'data_lidar.dat','Delimiter', ',');

a = p(1,1);
b = p(2,1);
x = 0: 0.01:1;
% ax + by = 1
y = (1/b) - ((a/b)*x);
figure
plot(x,y);
title('Pseudo inverse - Lidar')
axis([0 1 0 1]);
```

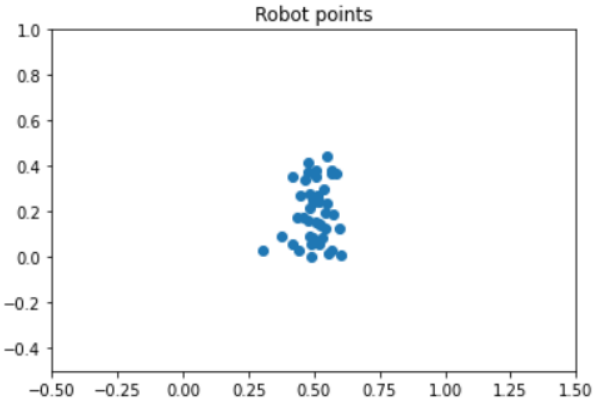
Then we use the data of the right wall (from 0 to 42) in the pseudo inverse function to get the following line as a result.



c. we save the previous point_x_y data in a .dat file, so that we can use it in python and implement the RASNAC algorithm on it.

First, we read and plot the inputs:

```
# reading robot's inputs
points=np.loadtxt('data_lidar.dat',delimiter=',')
plt.scatter(points[:,0], points[:,1])
plt.title('Robot points')
plt.xlim(-0.5,1.5)
plt.ylim(-0.5,1)
plt.show()
```



Number of iterations and tolerance are two variables, which we found through trial and error.

```
# these two numbers are based on trial and error
numberOfIterations = 10000
tolerance = 0.075
max_inlier = 0
optimal_coefficients = []
```

First, we choose two random points as the points on our line.

```
# Generating random indices
index1 = random.randint(0, len(points)-1)
index2 = random.randint(0, len(points)-1)
while index1 == index2:
    index1 = random.randint(0,len(points)-1)
```

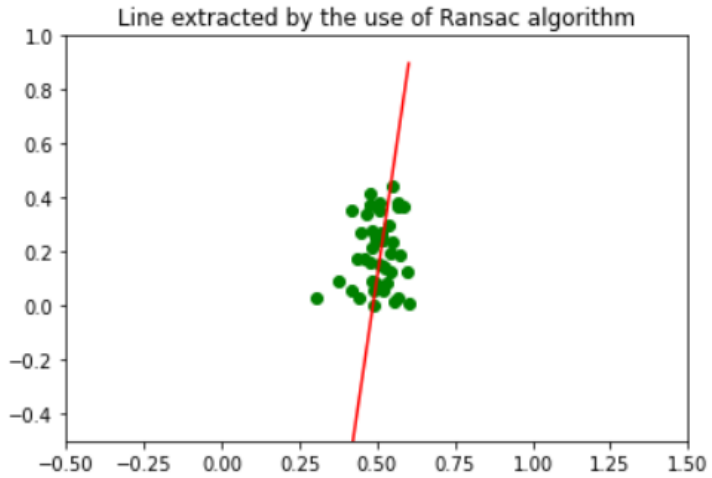
To find inlier points, we iterate through the rest of the points and find the distance of those points from our line. If its distance is less than the given tolerance, it is an inlier point. Afterwards, we count the number of the inlier points.

```
#find the coefficients for the line
#line equation : A*X + B*Y + C = 0   A = y1 - y2   B = x2 - x1   C = x1*y2 - x2*y1
x1 = points[index1][0]
x2 = points[index2][0]
y1 = points[index1][1]
y2 = points[index2][1]
A = y1 - y2
B = x2 - x1
C = x1*y2 - x2*y1

# finding inlier points
inlier = 0
for i in range(0, len(points)):
    # distance function =>  |Ax1 + By1 + C|/sqrt(A^2 + B^2)
    distance = abs(A*points[i][0] + B*points[i][1] + C)/(pow(A*A + B*B, 0.5))
    if distance < tolerance:
        inlier += 1
# better line is found
if inlier > max_inlier:
    optimal_coefficients = [A, B, C]
    max_inlier = inlier
```

We do this iteration for numberOfIteration times and find the maximum number of inlier points for each two randomly chosen points to create our optimal line.

Through this line and the x and y range, we plot the final line, which is the result of our algorithm.

```
# scatter the input values in green
plt.scatter(points[:,0], points[:,1], color="green")
A = optimal_coefficients[0]
B = optimal_coefficients[1]
C = optimal_coefficients[2]

# ploting the result in red based on the coefficients
x_vals = [point[0] for point in points]

x_range = [min(x_vals), max(x_vals)]
# line equation
y_range = [(-C-A*x_range[0])/B, (-C-A*x_range[1])/B]

plt.plot(x_range, y_range, color="red")
plt.title('Line extracted by the use of Ransac algorithm')
plt.xlim(-0.5,1.5)
plt.ylim(-0.5,1)
plt.show()
```



d. split and merge is a recursive algorithm in which we find the point with maximum distance and if it's greater than epsilon (we choose epsilon with trial and error), we split the points in half and call the function for each subset of points and save the result by merging these two results.

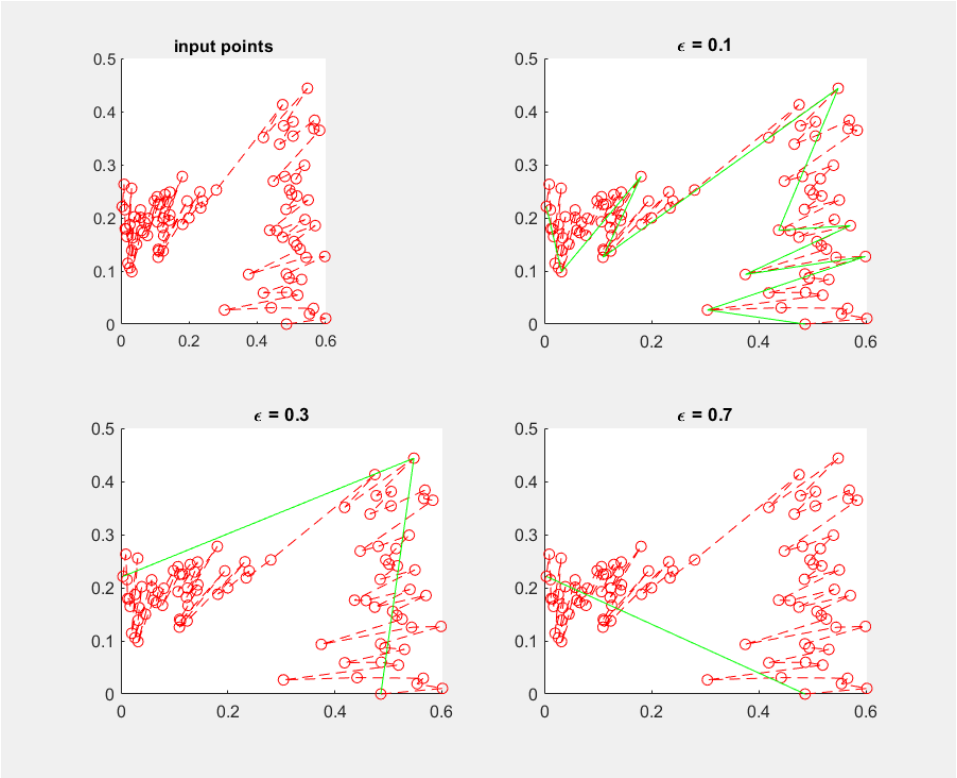
```
% Find the point with maximum distance
dmax = 0;
end_point = length(points);
for i = 2:end_point-1
    d = penDistance(points(:,i),points(:,1),points(:,end_point));
    if d > dmax
        index = i;
        dmax = d;
    end
end
% If max distance is greater than epsilon, recursively get the result
if dmax > epsilon
    % recursive call
    recResult1 = split_and_merge(points(:,1:index),epsilon);
    recResult2 = split_and_merge(points(:,index:end_point),epsilon);
    result = [recResult1(:,1:length(recResult1)-1) recResult2(:,1:length(recResult2))];
```

If the max distance is less than epsilon then it's one of the points of our final line.

```
else
    result = [points(:,1) points(:,end_point)];
```

You can see the result for epsilon = 0.1, 0.3, 0.7.

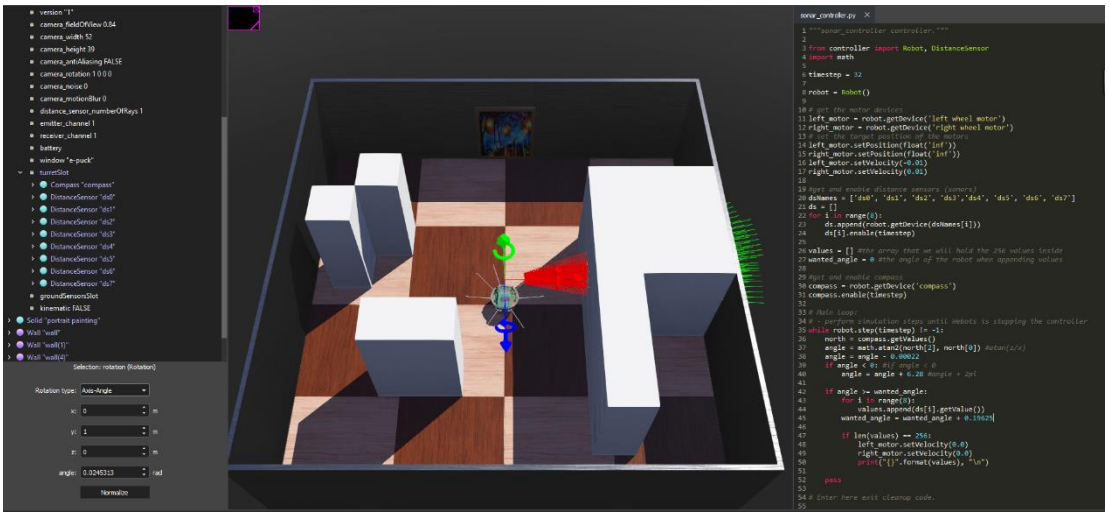
```
for epsilon = [0.1 0.3 0.7]
    result = split_and_merge(Points,epsilon);
    plt_place = plt_place + 1;
    subplot(2,2,plt_place);
    hold on
    plot(Points(1,:),Points(2,:),'--or');
    plot(result(1,:),result(2,:),'green');
    title(['\epsilon = ' num2str(epsilon)]);
end
```



Extra Points)

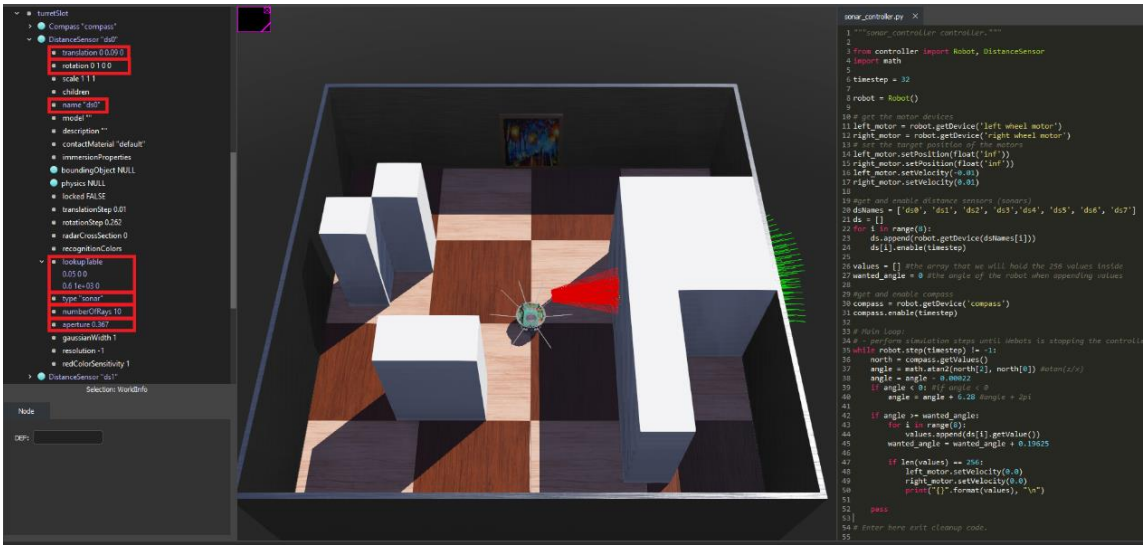
a. We need to add 8 distance sensors of type "sonar" and a compass, to our torretSlot.

sonar:



We rename the 8 sonars to ds0, ds1, ..., ds7. In our code, we can get distance sensors by the name ds0 to ds7 we gave it. We got an array with 256 entries, so the difference between the rotation angle of the sonars must be 2pi/256, which is equal to 0.0245 radian.

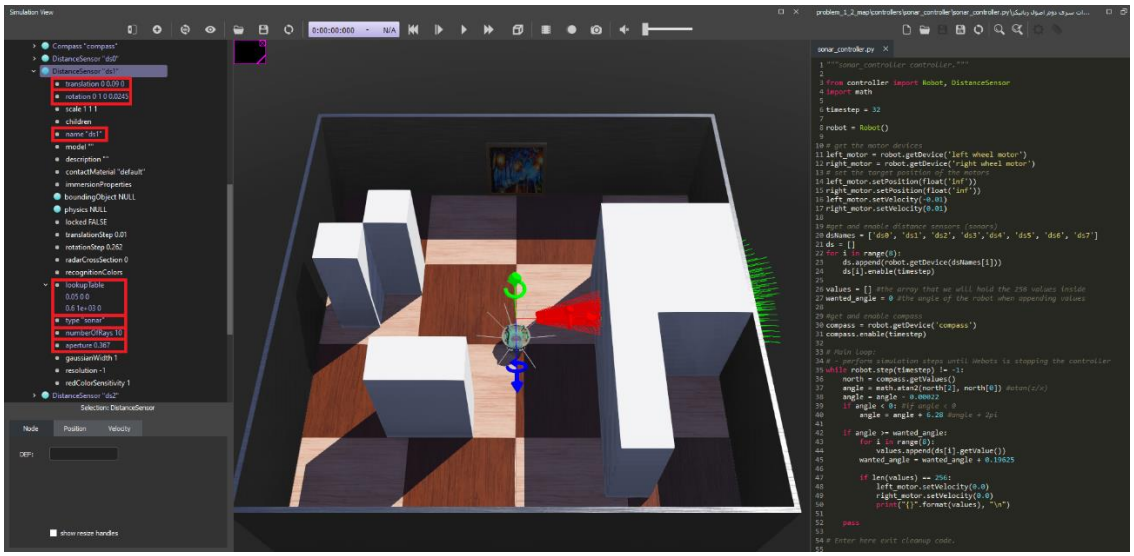
Ds0:



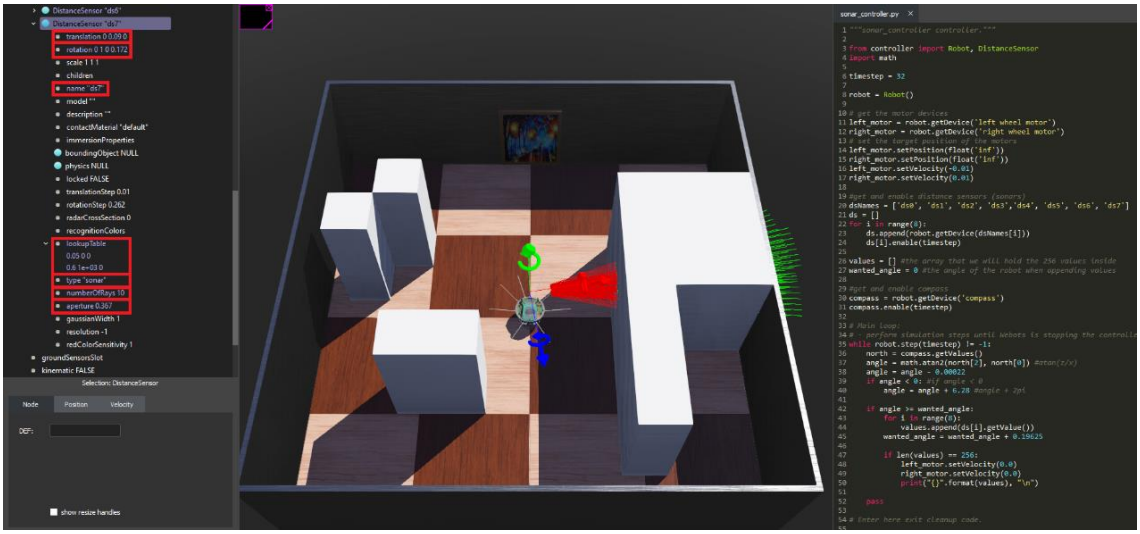
Lookup table works like a map. x is equal to the real distance of our sonar to the wall, y is equal to the distance our sonar sets and z is the noise. In our case, min x equals to 0.05 meters and min y is equal to 0 and we have no noise (z equals to 0). Max x is equal to 0.6, which maps to max y of 1000.

We need a compass, because we only have 8 distance sensors but we need 256 entries so our robot must rotate to get all these entries. Our rotation angle was 0.024 (as explained previously). For each sonar 0 to 7, the rotation angle is 0.024 radians more than the previous (rotation angle of dsi = i*0.024), until it is equal to 0.1717 in ds7. In the next rotation ds0 is like ds8 (ds1 like ds9, and so on, it repeats until ds255) so its angle must be 0.024 plus the previous angle of ds7 which is equal to 0.19625 (line 45 of the code). The angle must increase until we get our 256 samples (256/8 = 32 so we append the value of the 8 sonars to our array 32 times).

Ds1:

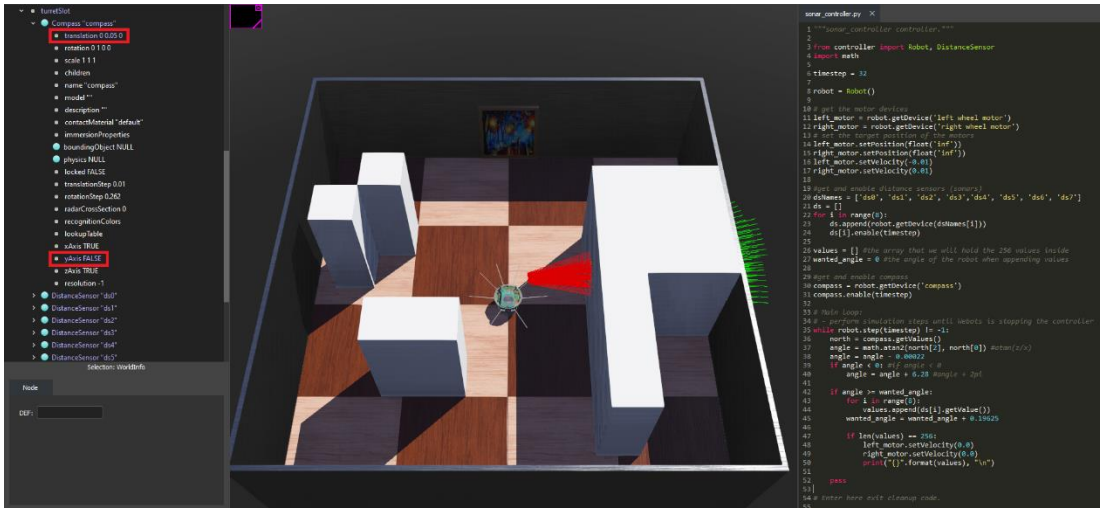


Ds7:



b. We give a very low velocity (0.01) to the motors so it rotates around itself slowly. We can get the angle of the robot with the compass each time (compass.getValue() returns a vector that indicates the north direction specified by the coordinateSystem field of the WorldInfo node). As we don't move along the y axis we disable the yAxis property (north[1] will be equal to NaN) and only get 0 and 2 for x and z axis. Then calculating arcTan of z/x gives us the angle.

Compass:

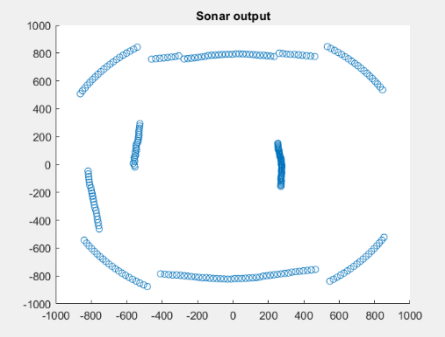


c. The first issue we faced here was that when the program runs for one time step, the robot turns a little. Therefore the first angle that we read from the compass isn't exactly 0 (it's about 0.00022), so we subtract this value from the angles that are read (line 35 of the code), or we could have set the initial value of "wanted_angle" to 0.0002. In fact, if we didn't do any of those, it would still be okay since it's such a small value compared to the next "wanted_angle" value. In addition, the robot's rotation speed needs to be very low so that we can read the values from the sonars when they are in the wanted position, with as little error as possible.
Another issue that we faced was that after reaching pi, the angle becomes less than zero and in order to prevent this, we add 6.28 to it.
In order to get the distance values, each time we compare the angle with the wanted angle, and if they are equal, the values of 8 distance sensors are added to our value array (In reality the angle and wanted_angle are never equal as there is a small amount of error so we use >= instead of ==). Then we add 8*0.024 to the wanted angle, which was initially 0 (It means we get the value of sensors each time the robot rotates for 8 times of 0.024, so all the distance sensors get new values).
When all 256 entries are read, the velocity of the robot's motors is set to 0 so that the robot stops spinning and then we print the final values.

d.

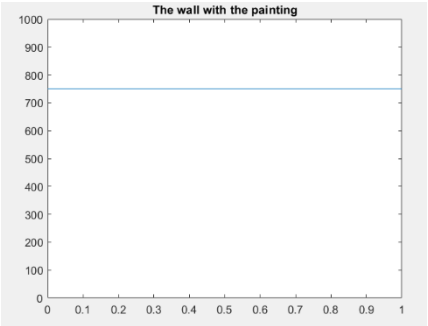
Points:

```
clear;clc;
%r = importdata('golabi_distance.txt', ',');
r = importdata('sonar.txt', ',');
X = [];
Y = [];
% x = r cos  y = r sin
for i = 1:256
   x1 = r(i)*cosd(1.41*i-1.41);
   X = [X, x1];
   y1 = r(i)*sind(1.41*i-1.41);
   Y = [Y, y1];
end

%scattering two walls
figure
scatter(X,Y);
title('Sonar output')
```

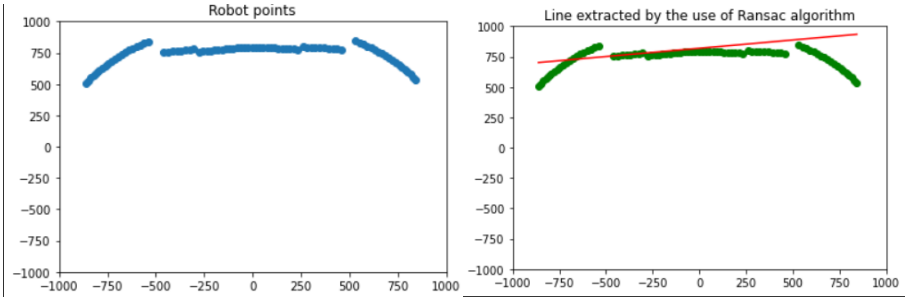


Pseudo:

```
% getting just one wall
X = X(24:107);
Y = Y(24:107);
Z = [X.',Y.'];
c = ones(size(X.'));
p = pseudo_inverse(Z, c);
disp(p);
writematrix(Z,'data_sonar.dat','Delimiter', ',');

a = p(1,1);
b = p(2,1);
x = 0: 0.01:1;
y = (1/b) - ((a/b)*x);
figure
plot(x,y);
title('The wall with the painting')
axis([0 1 0 1000]);
```

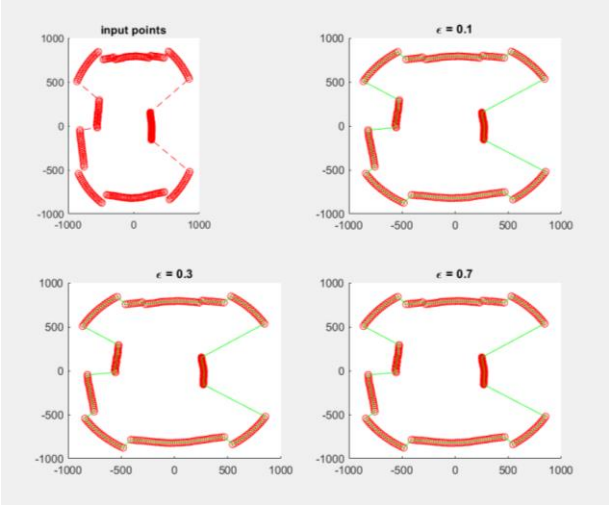


Ransac:



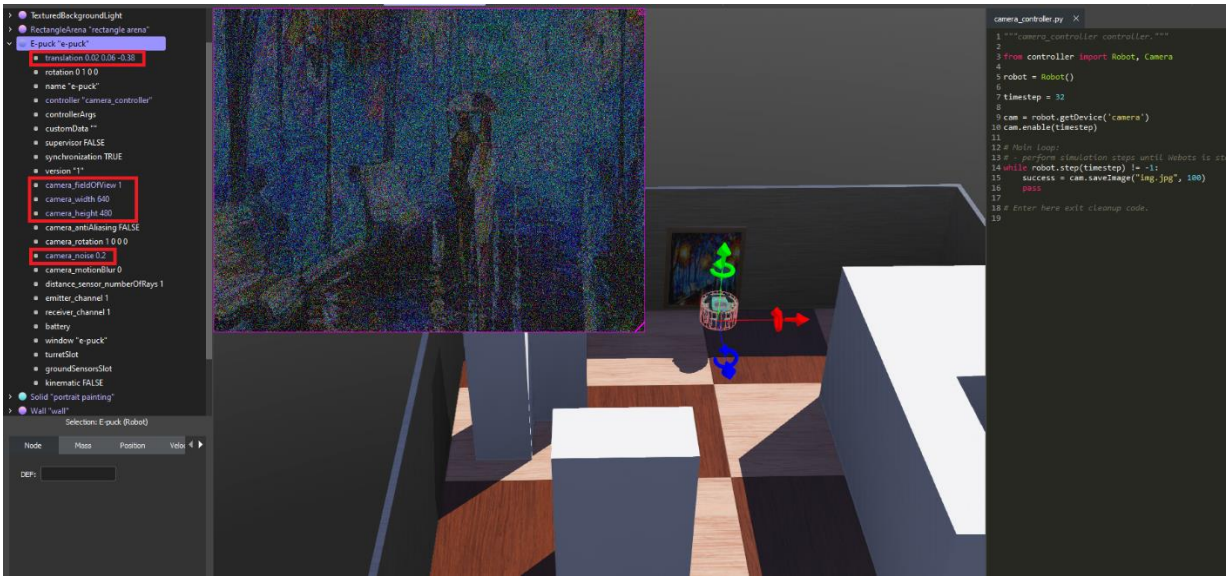
Split and merge:

```m
clc;
clear;
Points = importdata('data_sonar.txt',',');
Points = [Points(:,1).' ; Points(:,2).'];

%ploting input points
figure;
plt_place = 1;
subplot(2,3,plt_place);hold on
plot(Points(1,:),Points(2,:),'--or');
title('input points');
for epsilon = [0.1 0.3 0.7]
    result = split_and_merge(Points,epsilon);
    plt_place = plt_place + 1;
    subplot(2,2,plt_place);
    hold on
    plot(Points(1,:),Points(2,:),'--or');
    plot(result(1,:),result(2,:),'green');
    title(['\epsilon = ' num2str(epsilon)]);
end
```



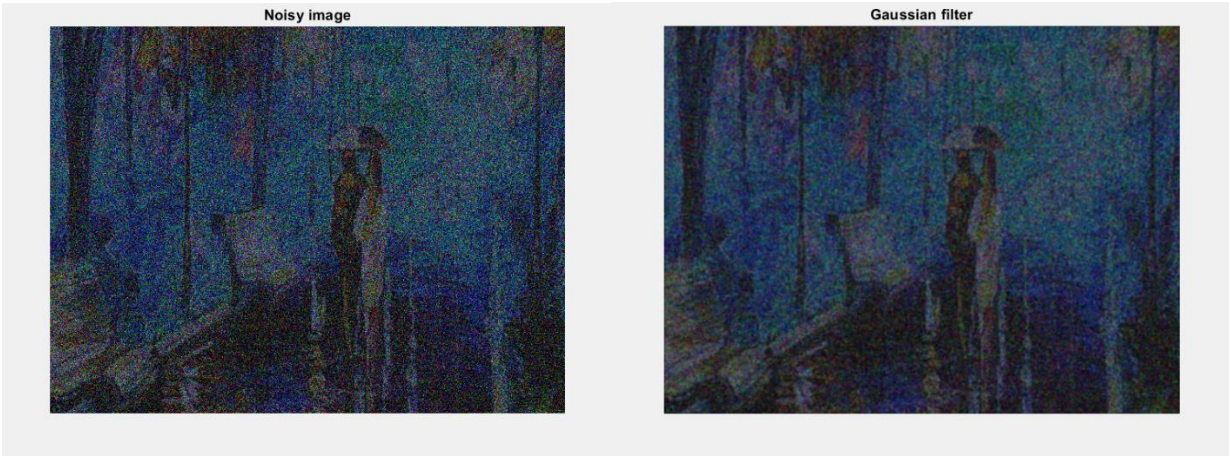
2)

We set height and width and noise of the camera and also set the robot in front of the painting by hand.



home > nanami > Downloads > C q2_filterGaussian.m

```
clear;
clc;
img = imread('img.jpg');
%img = imnoise(img, 'gaussian');
figure
imshow(img);
title('Noisy image')
sigma = 1;
kernel = zeros(5,5);
%sum of elements of kernel
sum = 0;
for i = 1:5
    for j = 1:5
        kernel(i,j) = exp(-1*((i-3)^2 + (j-3)^2)/(2*sigma^2));
        sum = sum + kernel(i,j);
    end
end

%normalizing kernel
kernel = kernel/sum;
[m,n] = size(img);
%outImg = padarray(img, [2,2]);
outImg = uint8(convn(img, kernel, 'same'));
figure
imshow(outImg);
title('Gaussian filter')
imwrite(outImg, 'out.png');
```



You can see the difference of the gaussian filter on the noisy and the noiseless image.

The right image is the output of the prewitt filter on the noisy image, while the left one is on the noiseless image.

home > nanami > Downloads > C q2_filterPrewitt.m

```
% output from the gaussian filter image
img = im2double(rgb2gray(imread('out.png')));

% masks
x=[-1 -1 -1;0 0 0;1 1 1]/6;
y=[-1 0 1; -1 0 1; -1 0 1]/6;
Gx=abs(conv2(img,y,'same'));
Gy=abs(conv2(img,x,'same'));
G = sqrt( Gx.^2 + Gy.^2);
out = G > 0.04; % Threshold
figure;
imshow(out);
title('Prewitt filter')
```