Exercise 2 – Monte Carlo Localization Using Landmarks

Ori Aharon 200274504

Nadav Lehrer 302170378

a) In this section we were asked to experiment with the two classes given i.e. cRobot and cWorld classes.
Initially we created an object of the class which initiates the world which the robot lives in, the after we initiated three objects of the class cRobot and plotted them into the world with the states:

$$(x, t, \theta): \{\ (40, 40, 0), (60, 50, \pi/2\ ), (30, 70, 3\pi/4)\ \}$$

see specific code for this section at the bottom of this document.

The results were as follows:

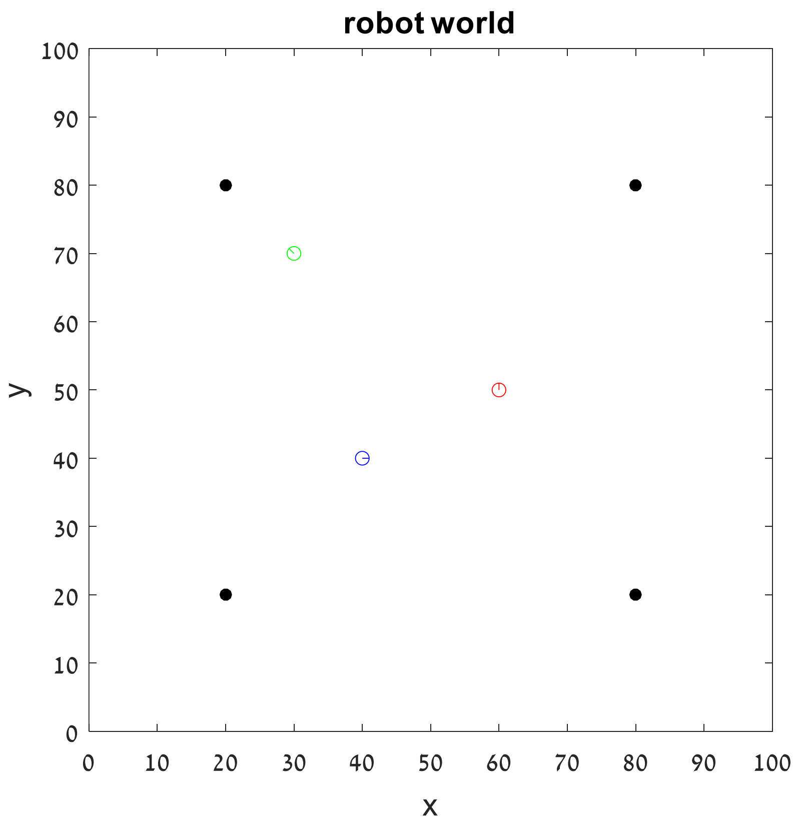


Figure 1: The three robots created (green, red and blue) plotted on the world map.

b) In order to move the robot (or a particle during the filtering process) as needed according to the model presented to us, a "move" function was asked to be added to the cRobot class.
To do so, we must first look at the model at a time step increment:

$$\theta_{t+1} = \theta_t + u_1$$
$$x_{t+1} = x_t + u_2 \cos(\theta_{t+1})$$
$$y_{t+1} = y_t + u_2 \sin(\theta_{t+1})$$

Where:

$$\begin{pmatrix} u_1\ [rad] \\ u_2\ [length] \end{pmatrix}$$

Are the action commands input of desired angle drive of u_1 and forward movement of u_2 . Of course, in real life (a scenario which we intend to imitate) the motor commands are corrupted by Gaussian noise, which are modeled with zero mean and standard deviations of $\sigma_1 = 0.1\ [rad]$ as the turn noise and $\sigma_2 = 5$ as the forward drive noise. Thus, by defining the appropriate random variables as:

$$\epsilon_1 \sim \mathcal{N}(0, \sigma_1)$$
$$\epsilon_2 \sim \mathcal{N}(0, \sigma_2)$$

We yield the model used in this assignment:

$$\theta_{t+1} = \theta_t + u_1 + \epsilon_1$$
$$x_{t+1} = x_t + (u_2 + \epsilon_2) \cdot \cos(\theta_{t+1})$$
$$y_{t+1} = y_t + (u_2 + \epsilon_2) \cdot \sin(\theta_{t+1})$$

Furthermore, it is assumed that the **world is cyclic** i.e. if the resultant $(x_{t+1}, y_{t+1}, \theta_{t+1})$ are beyond the map limits (which are specified in the cWorld class as 100 by 100) or the anlgle doesn't lie on the interval $[0, 2\pi)$, the robot will re-enter on the opposite side of the map with an angle in the appropriate interval. This was done with simple "if\else" statements as can be seen in the code at the bottom of the text.

c) As the robot moves it needs to collect the measurements in each time step. These measurements contain the distance from each landmark in the world map. The measurements are noisy and contain white Gaussian noise which is represented by a random variable:

$$\epsilon_{measurement} \sim \mathcal{N}(0, \sigma_m = 5)$$

To obtain the measurement model we simply calculate the Euclidian distance form each associated j^{th} landmark which position is denoted by (m_x^j, m_y^j):

$$d_t^j = \sqrt{\left(x_t - m_x^j\right)^2 + \left(y_t - m_y^j\right)^2} + \epsilon_{measurement}$$

Where d_t^j is the distance from the j^{th} object at time t.

This is the core calculation in the function "sense" which was added to the cRobot class

d) In this section we were asked to add a final function to the cRobot class. This function "measurement probability" returns the weight of a particle which stands at position $\left(p_x, p_y\right)_t$ position at time t. The inputs of this function are the location of a particle $\left(p_x, p_y\right)_t$, the measured distance of the robot at time t $(d_t^1, \dots, d_t^4)$ and the map object, which contains the landmarks. The distance r_t^j from a particle to an associated j^{th} landmark at time t was as follows:

$$r_t^j = \sqrt{\left(p_{x_t} - m_x^j\right)^2 + \left(p_{y_t} - m_y^j\right)^2}$$

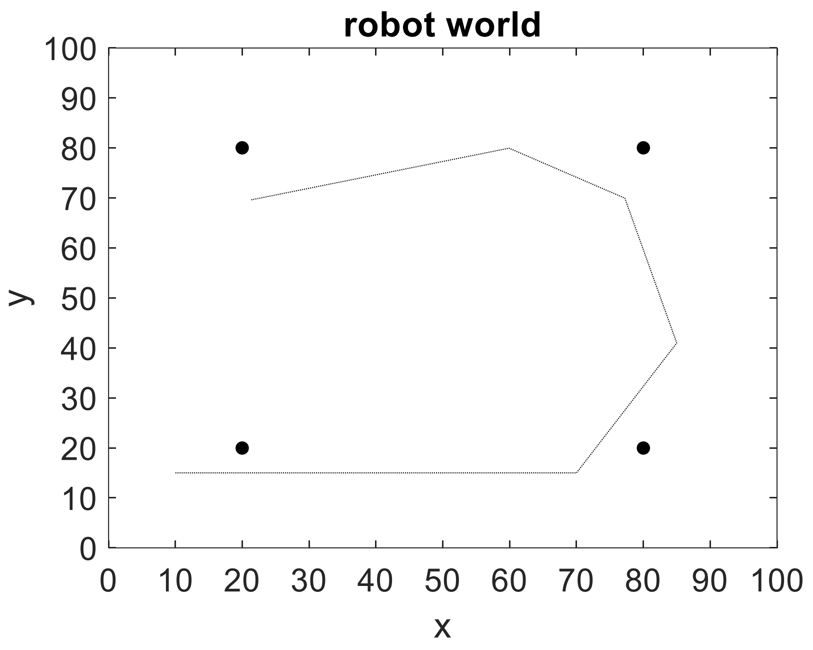
And the probability value of this r_t^j outcome will be as follows:

$$p_t^j = \frac{1}{\sqrt{2\pi\sigma_m}} e^{-\frac{\left(r_t^j - d_t^j\right)^2}{\sigma_m^2}}$$

As all the measurements $(z_1, .., z_4)_t$ are assumed to be i.i.d., the joint probability becomes a product of probabilities and so, the weight of a particle p with respect to the likelihood function, becomes:

$$p_t = P((z_1, .., z_4)_t | x_t, m) = \prod_{k=1}^{4} p_t^k$$

e) By utilizing the "move" function written with zero noise parameters, and inputting the



following commands:

$$\binom{u_1}{u_2}:\left\{\binom{0}{60},\binom{\pi/3}{30},\binom{\pi/4}{30},\binom{\pi/4}{20},\binom{\pi/4}{40}\right\}$$

We were able to plot the robot commands path:

Figure 2: the robot's commanded path.

f) In this section we once more utilized the "move" function, now setting the noise parameters to $\sigma_1 = 0.1\ [rad]$ as the turn noise and $\sigma_2 = 5$ as the forward drive noise, and yielded the graph:

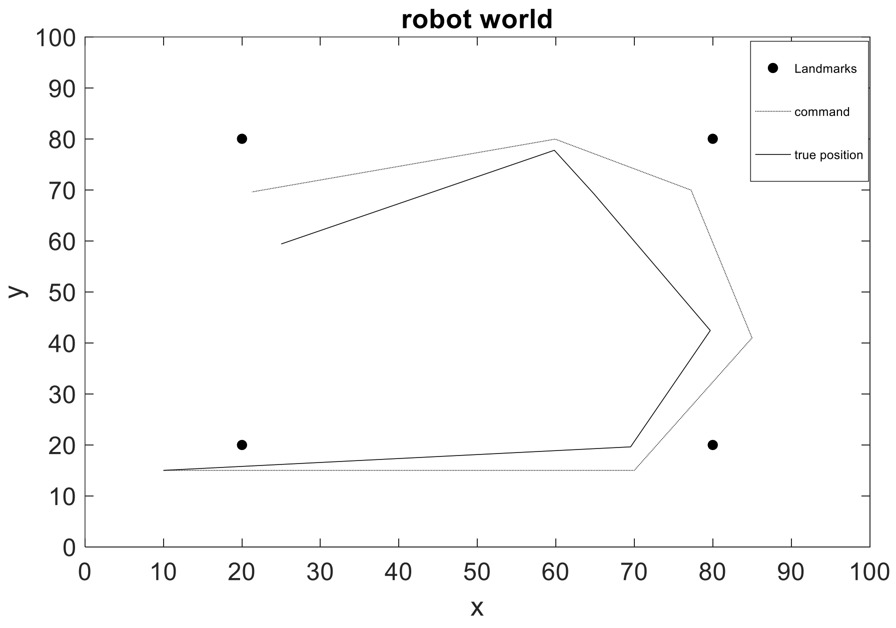


Figure 4: the robot's commanded path and true path.

g) Before showing our particle filter results, we will demonstrate the Low Variance Sampling function we wrote (according to the algorithm presented in Probabilistic Robotics book + weights normalization part). To see that this function works, we weighted equally distributed x samples according to a Gaussian and resampled these x's according to their weight. Then after, the histogram of the samples should return the Gaussian. The results are as follows:

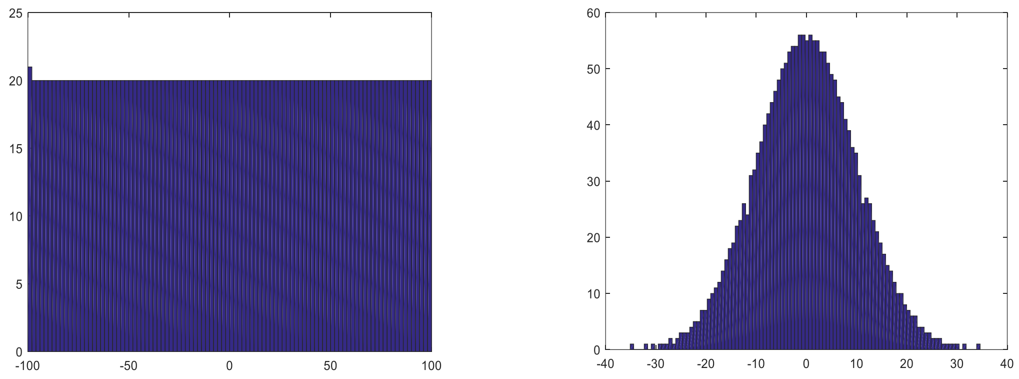


Figure 5: equally distributed samples on the left and re-sampled according to a Gaussian, on the right.

Next, we preformed the particle filter process to estimate the position of the robot. At each time step we projected the particles according to the action process, the after we weighted them and resampled. After each resampling, the weights of the particles are equal and so calculating the mean by a simple average will yield the position estimation. In the next figures

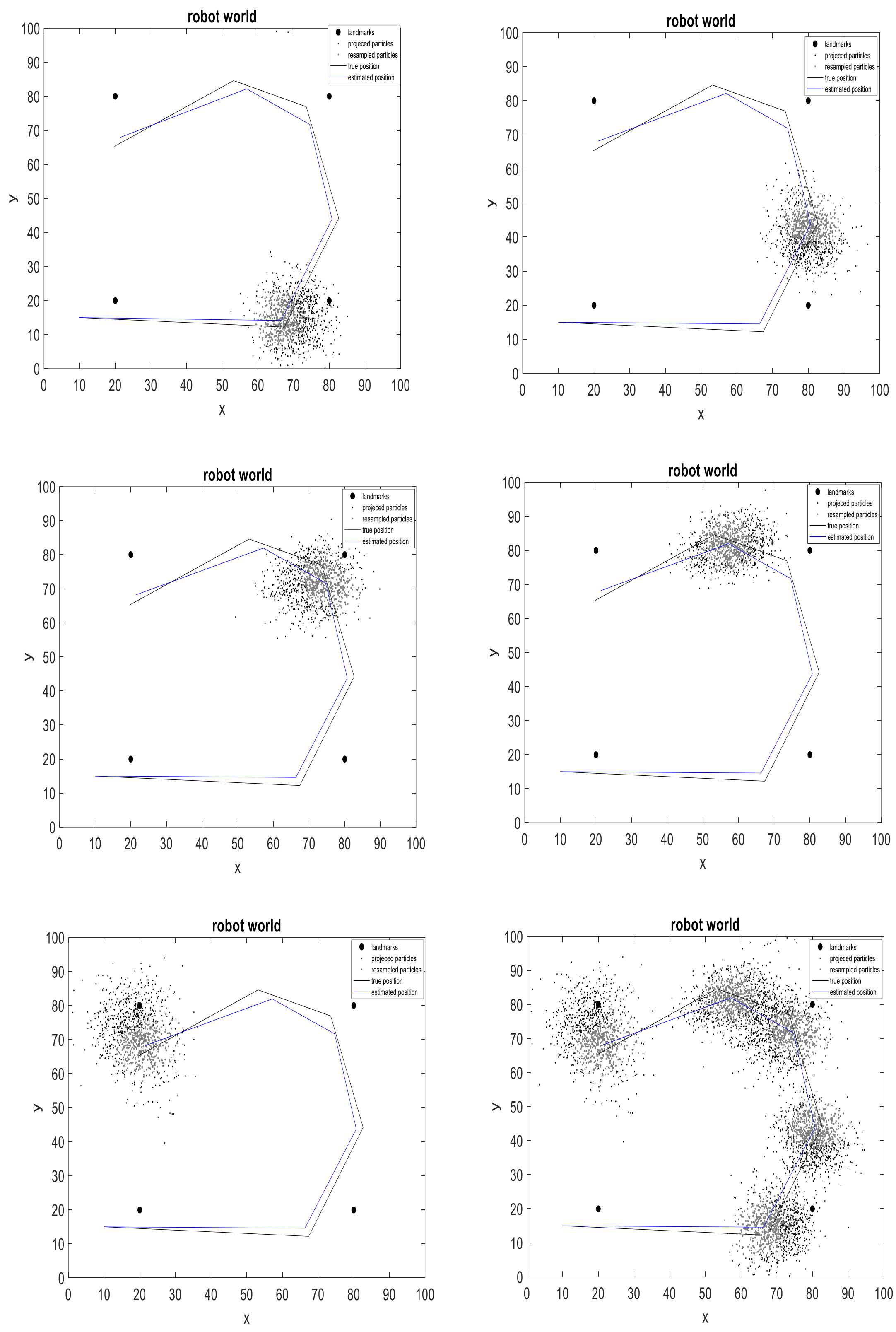


Figure 5: the estimation process. We can see the time step estimation at each time interval, from t=1 at the top left to t=5 at the bottom left. At the bottom right we can see the whole process. Notice that the resampled particle set in gray, is smaller in variance compared to the projected set in black.

Next we will demonstrate another simulation:

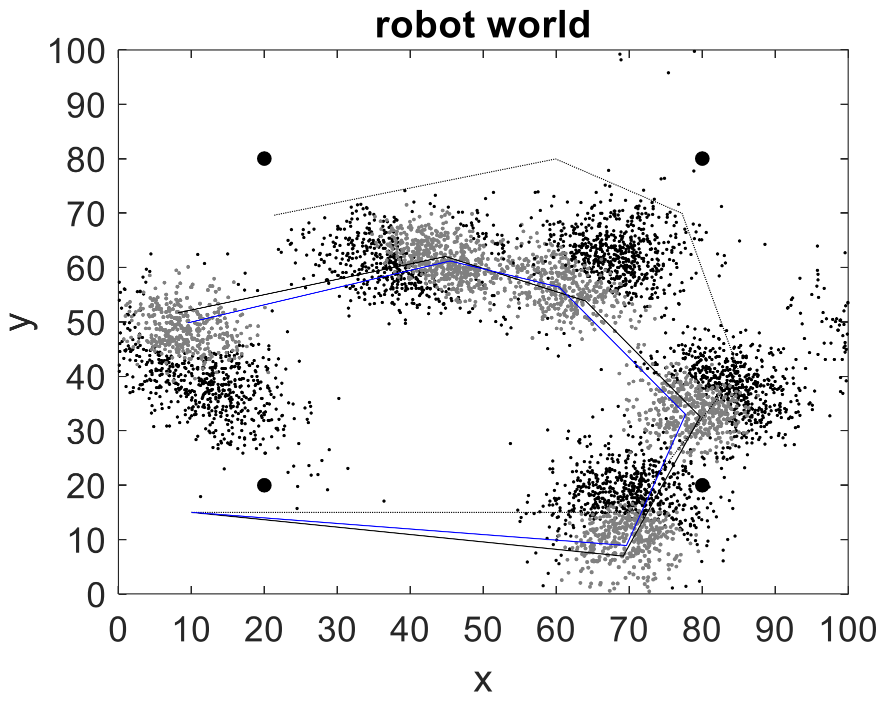


Figure 6: another random simulation

And one more, that will demonstrate that the particle filter works even if the robot was projected to the other side of the map, due to wold periodicity:

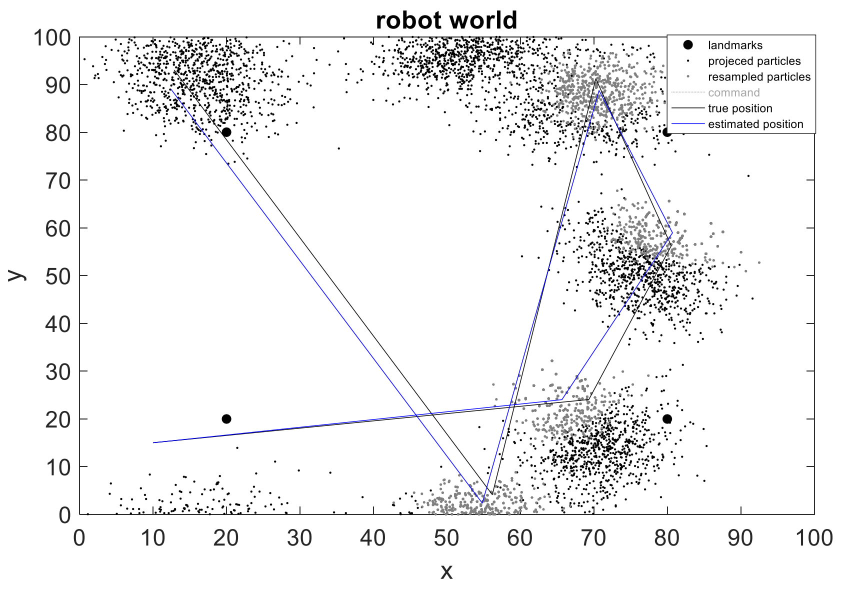


Figure 7: a simulation in which the robot re entered on the opposite side of the map at t=4, due to periodicity of the world map.

Another simulation with 10,000 particles yielded:

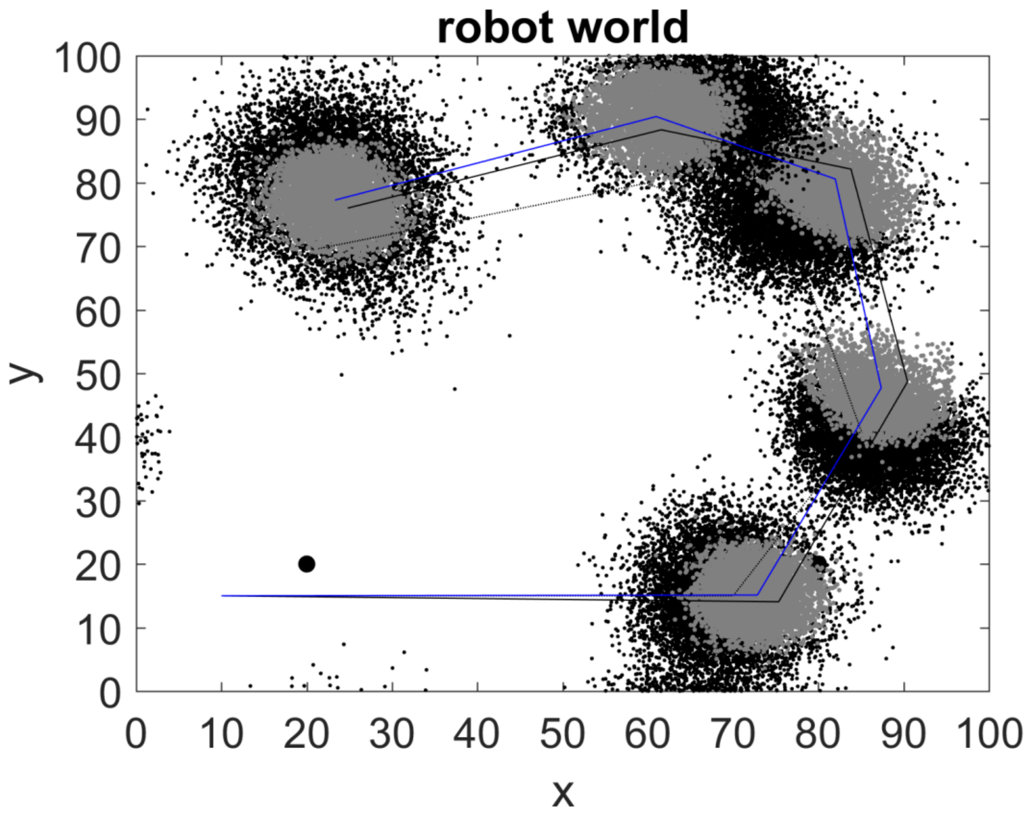


Figure 6: another random simulation, now with 10K particles

Appendix: Matlab Code

Section a -Experiment with the cRobot class

```
myworld=cWorld();
myworld.plot();
hold on;
myrobot1=cRobot();
myrobot1.set(40,40,0);
myrobot1.plot();
myrobot2=cRobot();
myrobot2.set(60,50,pi/2);
myrobot2.plot();
myrobot3=cRobot();
myrobot3.set(30,70,3*pi/4);
myrobot3.plot();
hold off
```

section b - add a function "move" to the cRobot class

```
         function   obj = move(obj,u_rotation,u_translation)
           obj.theta = obj.theta+u_rotation + mvnrnd(0,obj.turn_noise^2,1);
           obj.x =  obj.x + (u_translation +
mvnrnd(0,obj.forward_noise^2))*cos(obj.theta);
           obj.y = obj.y + (u_translation +
mvnrnd(0,obj.forward_noise^2))*sin(obj.theta);
           %cyclic world
           if obj.x>100
               obj.x=obj.x-100;
           elseif obj.x<0
               obj.x=obj.x+100;
           end
           if obj.y>100
               obj.y=obj.y-100;
           elseif obj.y<0
               obj.y=obj.y+100;
           end
           if obj.theta>2*pi
               obj.theta=mod(obj.theta,2*pi);
           elseif obj.theta<0
               obj.theta=-mod(obj.theta,2*pi)+2*pi;
           end
         end
```

Section c - add a function "sense" to the cRobot class

this function simulates the robot measurements, it calculates the robot true position to each landmark and adds white noise according to the measurement noise parameter. the output is an array (column vector) r of distances to an associated landmark with white noise.

```
function   [r]=sense(obj,map_obj)
       r=[];
       r=( (obj.x-map_obj.landmarks(:,1)).^2 ...
       +(obj.y-map_obj.landmarks(:,2)).^2  ).^0.5 ...
       +mvnrnd(zeros(1,4),(obj.sense_distance_noise^2)*eye(4),1)';
end
forward_noise=5;
turn_noise=0.5;
sensor_noise=5;
myrobot1.set_noise(forward_noise,turn_noise,sensor_noise);
```

Section d - add a function "measurement_probability" to the cRobot class

this function is for particle weithing. r_measured is a column vector of the measured features at time t, pose_x is the particle position vector, map_obj is the map object which contains the features measured (landmarks)

```
function   [p] = measurement_probability(obj, r_measured, pose_x, map_obj)
       r_at_pose_x=( (pose(1)-map_obj.landmarks(:,1)).^2 ...
       +(pose(2)-map_obj.landmarks(:,2)).^2  ).^0.5;
   % beacuse the measurment of each landmark is an i.i.d random
   % variable, the joint probability of the measurements is the
   % product of probabilitys. we evaluate the the weight of the
   % particle accordind to the difference between the measured
   % distance to a landmark and the particle's distance to it.
       p=prod(normpdf(r_at_pose_x-r_measured,0,obj.sense_distance_noise));
end
```

Section e - intended robot path

```
myworld=cWorld();
myrobot=cRobot();
forward_noise=0;
turn_noise=0;
sensor_noise=5;
myrobot.set_noise(forward_noise,turn_noise,sensor_noise);
myrobot.set(10,15,0);
u=[0 60; pi/3 30; pi/4 30; pi/4 20; pi/4 40];

robot_command_pose=zeros(length(u)+1,2);
robot_command_pose(1,:)=[myrobot.x myrobot.y];
for i= 1:length(u)  % move the robot
    myrobot.move(u(i,1),u(i,2));
    robot_command_pose(i+1,:)=[myrobot.x myrobot.y];
end
```

```
myworld.plot();
hold on;
plot(robot_command_pose(:,1),robot_command_pose(:,2),'k:');
```

Section f

set the robot simulation position and noise parameters and move the robot according to the model. in each step take the measurements to the landmarks

```
forward_noise=5;
turn_noise=0.1;
sensor_noise=5;
myrobot.set_noise(forward_noise,turn_noise,sensor_noise);
myrobot.set(10,15,0);
u=[0 60; pi/3 30; pi/4 30; pi/4 20; pi/4 40];

robot_true_pose=zeros(length(u)+1,2); %initialize the robots true pose array
robot_true_pose(1,:)=[myrobot.x myrobot.y]; %the robot's first position
measurments=[];
for i= 1:length(u)  % move the robot
    myrobot.move(u(i,1),u(i,2));
    robot_true_pose(i+1,:)=[myrobot.x myrobot.y];
    measurments(:,i)=myrobot.sense(myworld); %take the measurement each step
end
plot(robot_true_pose(:,1),robot_true_pose(:,2),'k-');
```

section g - particle filtering

```
N=1000;
u=[0 60; pi/3 30; pi/4 30; pi/4 20; pi/4 40];
T=length(u)+1; %the last time steps occures after the last motor command

forward_noise=5;
turn_noise=0.1;
sensor_noise=5;

particle_weight=ones(N,1)*(1/N); % at the beggining each particle weighs the same

%initialize particles
particle=cRobot();
particle.set_noise(forward_noise,turn_noise,sensor_noise);
particle.set(10,15,0);

% it's more efficient memory, and run time-wise to put the particles into an array
particle_array=[particle.x*ones(N,1), particle.y*ones(N,1), zeros(N,1)];

% initialize the estimated position array
% the initial position is deterministic
robot_estimated_pose(1,:)=[10 15];

myworld.plot();
hold on;
% particle filter
for t=2:T % the estimation is from t=2 (pose in t=1 is known), to
```

```
    %advance particles with model and weight them
    for i=1:N
        %set each particle position to the previous position
        particle.set( particle_array(i,1),particle_array(i,2),particle_array(i,3));
        %move each particle
        particle.move(u(t-1,1),u(t-1,2));
        particle_array(i,:)=[particle.x particle.y particle.theta];
        particle_weight(i)=particle.measurement_probability(measurments(:,t-
1),particle_array(i,1:2),myworld);
    end
% % uncomment to show projected particles:

plot(particle_array(:,1),particle_array(:,2),'ko','MarkerSize',1,'MarkerFaceColor','k'
)

    % re-sample particles and estimate robot positiom via the expectation
    % of the re-sampled set:
    % Re-Sample
    particle_array=LoVarResampling(particle_array,particle_weight);
% % uncomment to show resampled particles:
    plot(particle_array(:,1),particle_array(:,2),'.','color',[0.5 0.5 0.5]);
%      pause();

    % calc the mean (expectancy of equally weighted particles)
    % of the particles position:
    robot_estimated_pose(t,:)=[mean(particle_array(:,1)) mean(particle_array(:,2))];
    %the Re-Sampled set is equally weighted (this opperation is
    %unnescesary but theoretically right):
    particle_weight=ones(N,1)*(1/N);

end
%ploting:

plot(robot_command_pose(:,1),robot_command_pose(:,2),'k:');
% legend('command')
plot(robot_true_pose(:,1),robot_true_pose(:,2),'k-');
% legend('true position')
plot(robot_estimated_pose(:,1),robot_estimated_pose(:,2),'b-');
legend('landmarks', ... %'command','true position','estimated position');
                    'projeced particles','resampled particles', ...
                    'projeced particles','resampled particles', ...
                    'projeced particles','resampled particles', ...
                    'projeced particles','resampled particles', ...
                    'projeced particles','resampled particles', ...
                    'command','true position','estimated position');
```

Low Variance Re-Sampling

The function used for re-sampling was written according to the algorithm presented in Probabilistic Robotics, with the added feature of weights normalization.

```
function equally_weighted_set=LoVarResampling(particle_set,weights)
%Example: mu = 0; sigma = 10; pd =makedist('Normal',mu,sigma);
%x=-100:0.1:100; weights=pdf(pd,x);re_sampled_x=LoVarResampling(x,weights);
%hist(re_sampled_x);

    % chi (2D array) is are the particles weighted according to the likelyhood pdf,
first
    % we need to know the number of particles N, and the state vector's
    % dimensions 'dim'(for example position velocity and temperature). first
    % let's assume that N>>dim and turn chi to N-by-dim (if not already)
    [n,m]=size(particle_set);
    N=max(n,m);        % particle number
    if N ~= n          % make 'chi' an N-by-dim array
        particle_set=particle_set';
    end
    clear n m
    % we need to normalize the weights so that cmd<=1
    weights=weights/sum(weights);

    cmd=weights(1);          % comulative mass dist. of weights
    i=1;
    equally_weighted_set=[];
    r=unifrnd(0,1/N);
    for m=1:N
        U = r+(m-1)/N;
        while U>cmd
            i=i+1;
            cmd=cmd+weights(i);
        end
        equally_weighted_set(m,:)=particle_set(i,:);
    end
end
```