Contents

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



\subsection*{section b - add a function "move" to the cRobot class}

\begin{verbatim}
        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_arra
    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
```