CENG 483 – BEHAVIORAL ROBOTICS

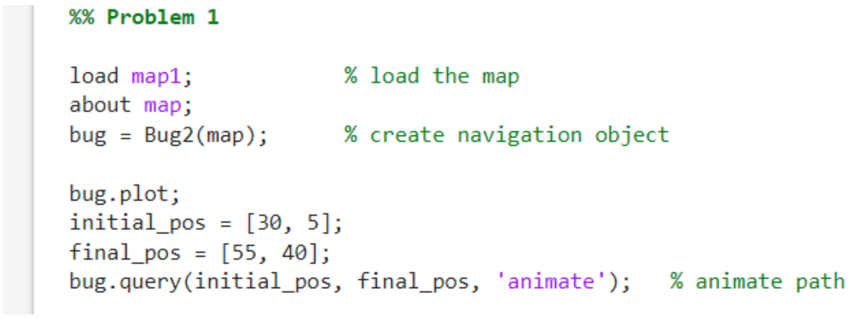
Homework Set #7   Due : 11.12.2023

Name : Cem Tolga MÜNYAS

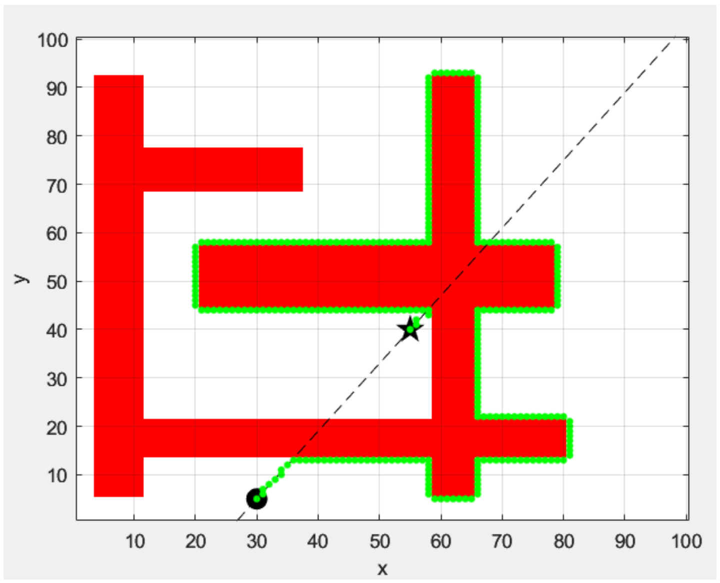
Student ID : 270206042

1. On the problem of Map Based Planning (pp. 91), use new initial point (30,5) and final goal as (55,40). Discuss the resulting navigation of the robot. Justify your answer.

In this question, we will be using the map-based planning approach. I created a 100x100 matrix field with obstacles with the "load map1" command. Then, I could achieve the information about the map with the command "about map". After that, I created a navigation object which is a bug in this case. We want to find the path that starts from the position (30, 5) and ends in the position (55, 40). With "bug.query" command, we animated the path of bug. The bug chose this way because it tends to choose the counterclockwise direction in every turn it takes.



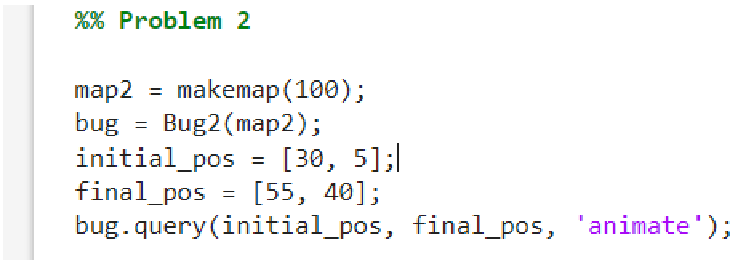
```
%% Problem 1

load map1;              % load the map
about map;
bug = Bug2(map);        % create navigation object

bug.plot;
initial_pos = [30, 5];
final_pos = [55, 40];
bug.query(initial_pos, final_pos, 'animate');   % animate path
```



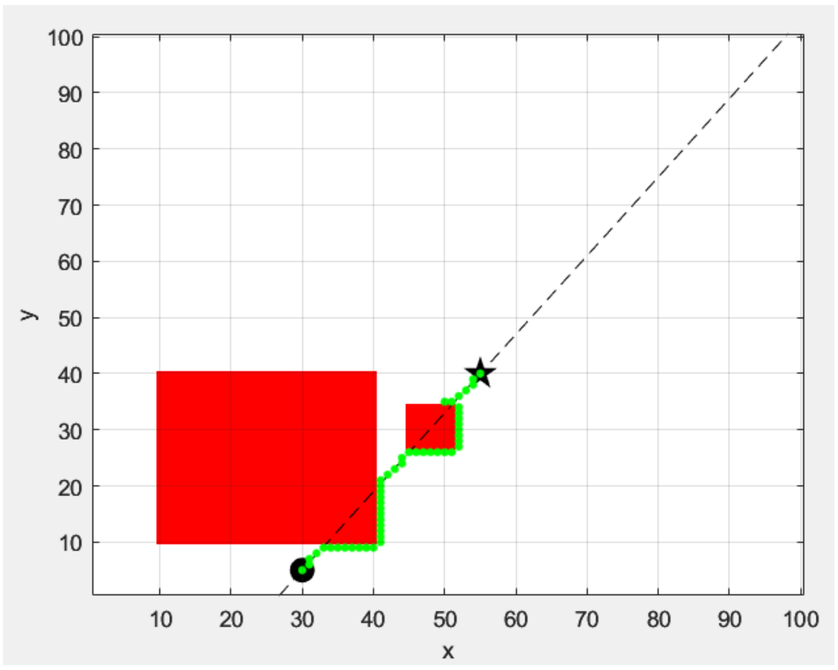
2. On the problem of Map Based Planning (pp. 91), use same initial points as in Problem 1, creates two different obstacles of your choice, i.e a circle centered at (x,y) with 10 meters diameter, and a polygon placed at (x_i,y_i), i = 1:4. Discuss the resulting navigation of the robot. Justify your answer.

In here, we were asked to create our own map using the toolbox map editor "makemap". So, with the command "mapmake(100)", we could create our own unique map by creating its obstacles in different shapes such as circles, polygons, or rectangles. We will be using the same starting point and the goal position in Q1.

I created the polygon placed at the endpoints (10,10), (10,40), (40,10), (40,40). Unfortunately, I could not create the circle due to some toolbox problem in my computer. It did not recognize the circle function. Instead of this circle, I created another polygon in the way of the goal. It basically does the same thing by following the counterclockwise directions in turns. The bug reached the goal in shorter distance by encountering the obstacles that we created.



```
%% Problem 2

map2 = makemap(100);
bug = Bug2(map2);
initial_pos = [30, 5];
final_pos = [55, 40];
bug.query(initial_pos, final_pos, 'animate');
```

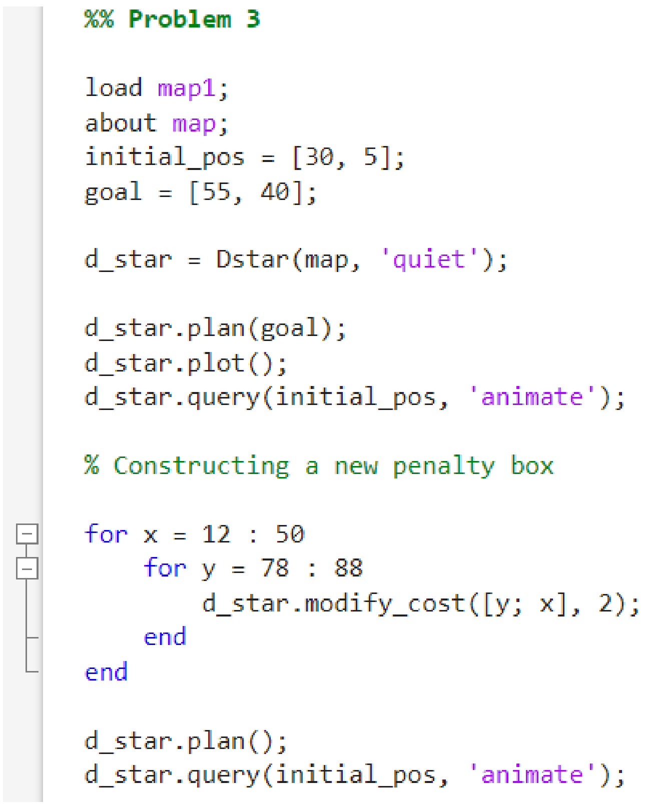


3. On the problem of D* (pp. 95), use new initial point as (30,5) and final goal as (55,40). Construct a new penalty box at 12 ≤ x ≤ 50 and 78 ≤ y ≤ 88. Discuss the resulting navigation of the robot, as compared with Problem 1. Justify your answer.

Firstly, we loaded the map1 and checked its info. Elements of the map are converted to transmittance values of robot. An obstacle has an infinite transmittance, and an empty space has nominal transmittance of 1. Then, using the D* algorithm, a robot was created. D* algorithm finds the path which minimizes the total cost of travel. After that, starting point and the goal position are set. In order to visualize the map and path finding of D*, we used *d_star.plot()* command. We also constructed a new penalty box which contains 12 ≤ x ≤ 50 and 78 ≤ y ≤ 88 as you can see in the corresponding code.

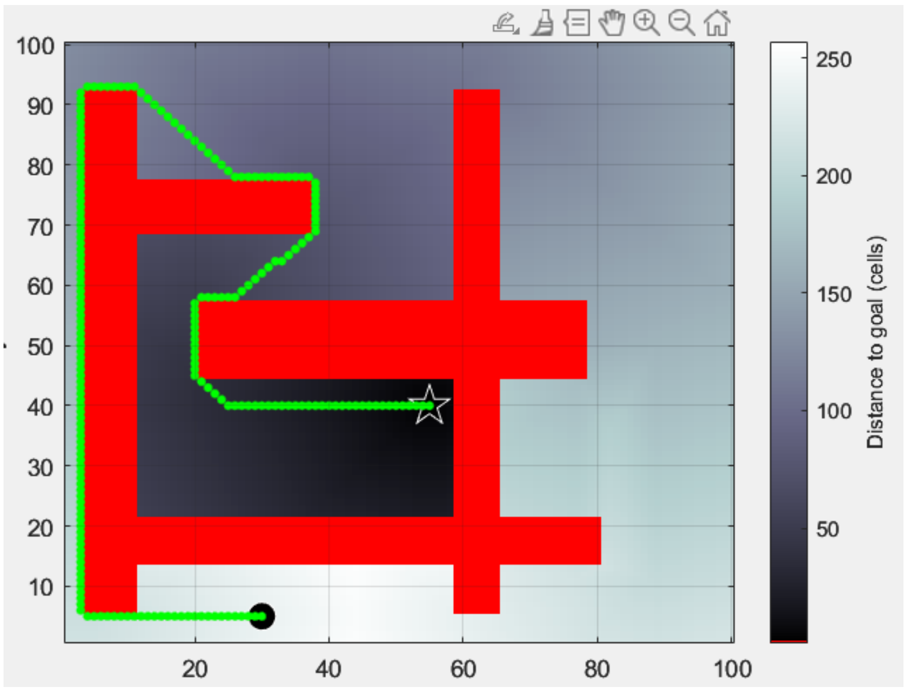
Since the D* algorithm uses the minimal cost for travel, we modified the cost for this penalty box. Finally, with the *d_star.query* command we animated the path of robot.

Here is the corresponding code for applying D* path finding algorithm.



```
%% Problem 3

load map1;
about map;
initial_pos = [30, 5];
goal = [55, 40];

d_star = Dstar(map, 'quiet');

d_star.plan(goal);
d_star.plot();
d_star.query(initial_pos, 'animate');

% Constructing a new penalty box

for x = 12 : 50
    for y = 78 : 88
        d_star.modify_cost([y; x], 2);
    end
end

d_star.plan();
d_star.query(initial_pos, 'animate');
```

Here is the visual for this process.



4. On the problem of Estimating Pose (pp. 113), use initial point at (0,0) and use 1200-time steps. Compare the results with 1000-time steps. Justify your answer.

In this question, we aim to estimate a pose based on the previous one. Here, I chose to use the Vehicle class which simulates the bicycle model. First of all, I created the odometry covariance per time step then created the vehicle with this noisy odometry. We considered the speed of vehicle as 1.5 m/s and steer angle of 0.5 rad. Then we predicted the next state based on the odometry values. After that, we attached a driver to the vehicle and generated a random path to direct the vehicle within a specified region. For the final process, we generated an ekf variable which uses "Extended Kalman Filter" and uses parameters as vehicle, odometry and initial covariance. When everything is done, we run the simulation for 1000 and 1200 times and plotted the corresponding figures for true vehicle path versus estimated path.

I also plotted the uncertainty ellipses for both 1000 times and 1200 times to observe the difference.

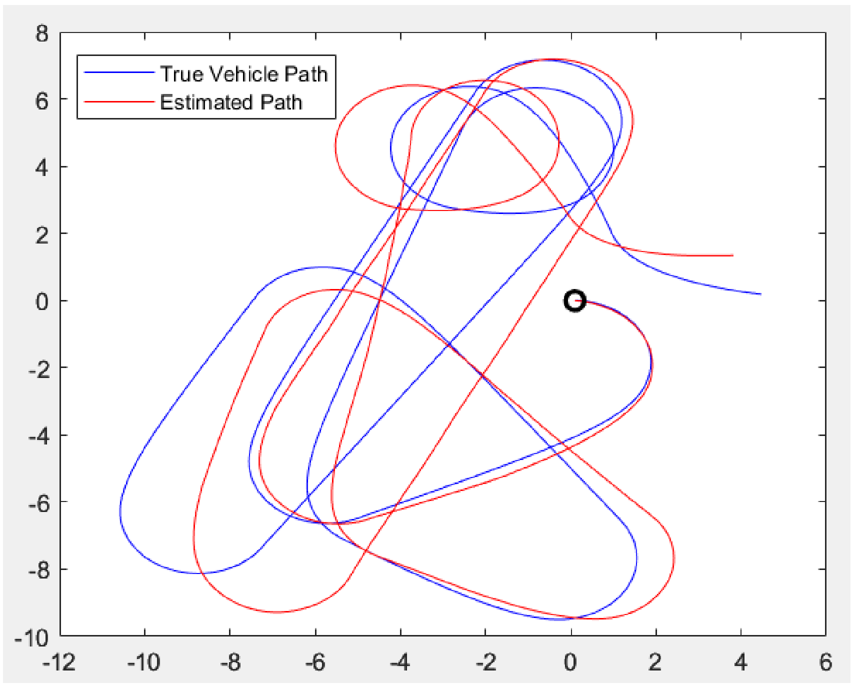
Here is the code for all of these processes and explanations which are commented out.

```
%Estimating the pose based on the previous one

% Odometry covariance per time step
V = diag([0.005, 0.5 * pi / 180].^2);

% Create a vehicle with noisy odometry
veh = Bicycle( 'covar', V );

% Apply a speed 1.5 m/s and steer angle 0.5 rad
odometry = veh.step(1.5, 0.5);

% Predict next state based on odometry
veh.f([0, 0, 0], odometry);

% Attach a driver object to this vehicle
veh.add_driver( RandomPath(10) );

% Initial Covariance
P0 = diag([0.005, 0.005, 0.001].^2);

% Extended Kalman Filter
ekf = EKF(veh, V, P0);

% Run the simulation for 1000 or 1200 times
ekf.run(1200);
```

Here is the code for plotting the true vehicle path vs estimated path.

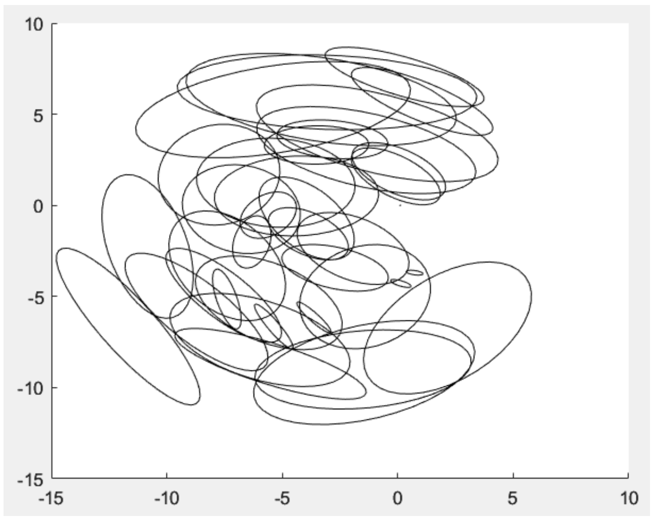
```
% Plot the true vehicle path vs estimated path
figure(1);
veh.plot_xy("b");
hold on;
ekf.plot_xy("r");
legend("True Vehicle Path", "Estimated Path");
```

Here is plot for these paths for run time 1200.

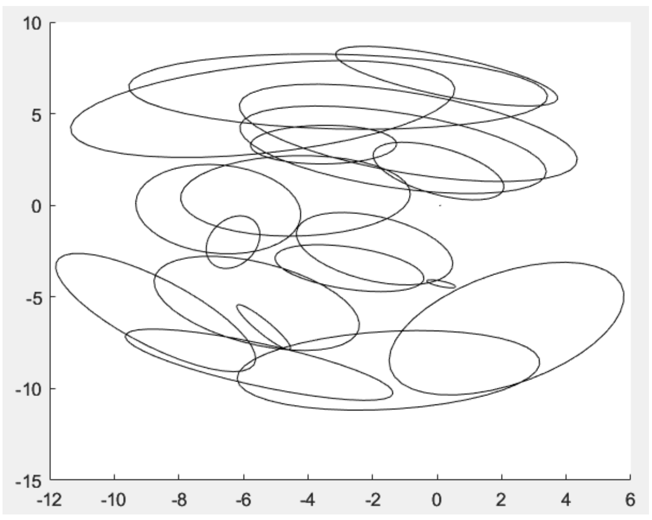


The code for demonstrating the uncertainty ellipses is shown below.

```
% Plot the uncertainity ellipses
figure(2);
ekf.plot_ellipse();
```



Ellipses for 1000 simulation time

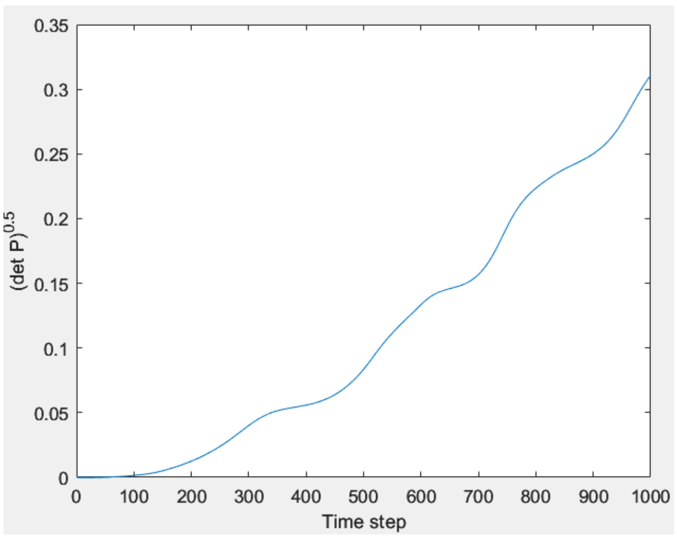


Ellipses for 1200 simulation time

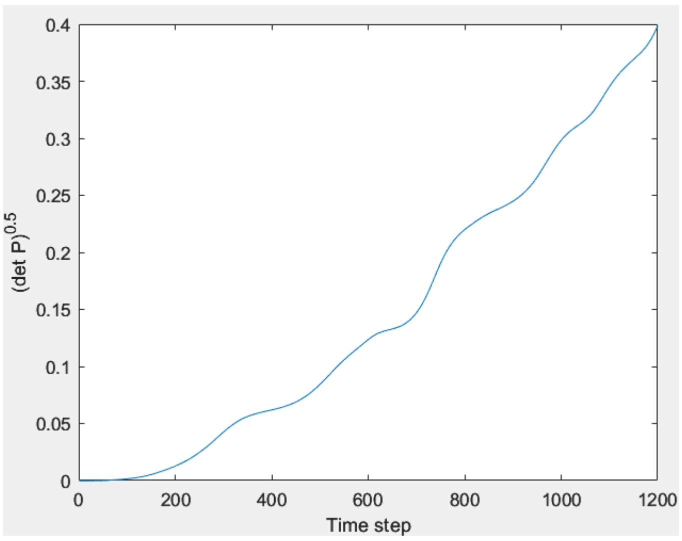
We can clearly see the difference between two simulation results that, ellipses have enlarged. Larger ellipses mean larger uncertainties.

For the final duty for this problem, we plot the covariance against time in order to observe the overall uncertainty. The corresponding code and figures for different run times are shown below.

```
% Plot the covariance against time
figure(3);
clf
ekf.plot_P();
```



Overall uncertainty for run time 1000



Overall uncertainty for run time 1200

We can clearly determine from the figures that overall uncertainty has increased as we increased the filter step.

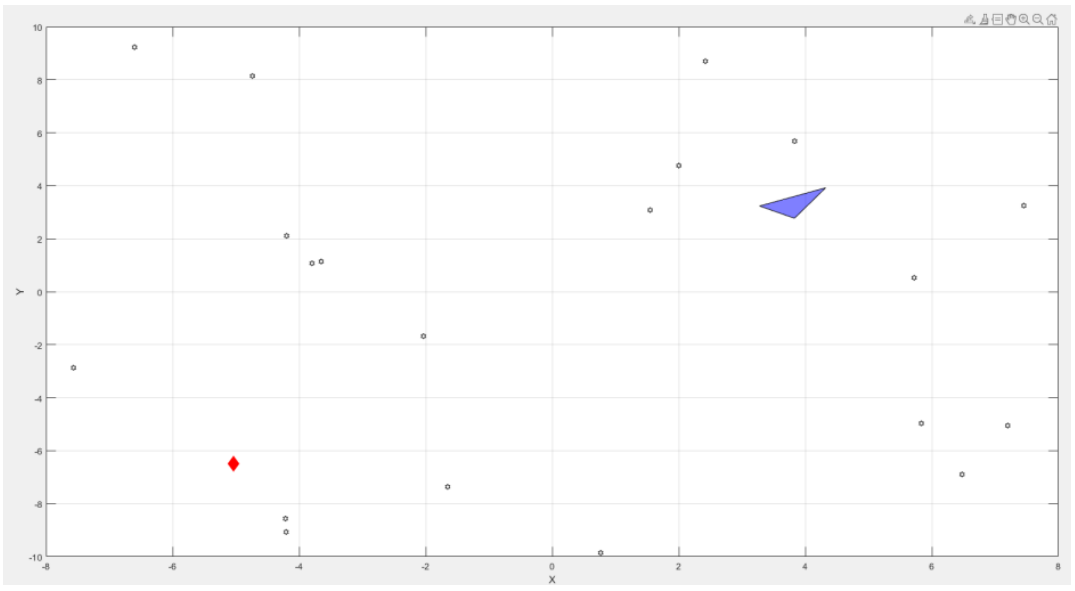
5. On the problem of Using a Map (pp. 116), use the same parameters as in Problem 4 and discuss the effects of Extended Kalman Filter.

Brief Description : The **EKF** object updates its state at each time step, and invokes the state update methods of the vehicle object. The complete history of estimated state and covariance is stored within the **EKF** object.

From starting position to the goal, the operation could not be implemented because the robot keeps moving away from the path. We follow the same steps in Problem 4 and upgrade it by creating a sensor that uses the map and vehicle state to estimate the landmark range and bearing with covariance. Then the Kalman filter is used with the parameters (estimated covariances, and initial vehicle state covariance P0). We run the simulation to observe the effects of changes. Here is the output for these processes. Additionally, MATLAB .m file is uploaded to the LMS in order to show my work.

```
% Odometry covariance per time step
V = diag([0.005, 0.5 * pi / 180].^2);

% Create a vehicle model with odometry covariance V
veh = Bicycle(V);

% Add a driver to it
veh.add_driver( RandomPath(20, 2) );

% Create a map with 20 point landmarks
map = LandmarkMap(20);

% Create a sensor that uses the map and vehicle state to estimate
% landmark range and bearing with covariance W
W = diag([0.1, 1*pi/180].^2);
sensor = RangeBearingSensor(veh, map, W);

% The Kalman Filter with estimated covariances V_est and W_est
% and initial vehicle state covariance P0
ekf = EKF(veh, V, P0, sensor, W, map);

% Run the simulation for the 1000 time steps
ekf.run(1200);

% Plot the true vehicle path and the estimated path
figure(1);
veh.plot_xy("b");
hold on;
ekf.plot_xy("r");
legend("True Vehicle Path", "Estimated Path");
```

Here is the proof that our robot can not keep the estimated path.



REFERENCES

- ***Robotics, Vision & Control, Peter Corke, Springer, 2011.***