CENG483: Behavioural Robotics
Izmir Institute of Technology
Fall 2023
Instructor: Dr. Berat Alper Erol

Homework Set: 07

Ozan Gülbaş

This homework answers the problem set sequentially.

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.*
   Firstly, we need to know how to use necessary toolbox functions to simulate the same environment as in the example. To get more information we went to the **DXform** source code. There we found an example code explaining each step. With the help of the example code we generated the needed simulation as follows:

```
load map1             % load map
goal = [55,40];       % goal point
start = [30, 5];      % start point
dx = DXform(map);     % create navigation object
dx.plan(goal)    % create plan for specified goal
dx.query(start, 'animate')   % animate path from this start location
```

   The resulting simulation plot:

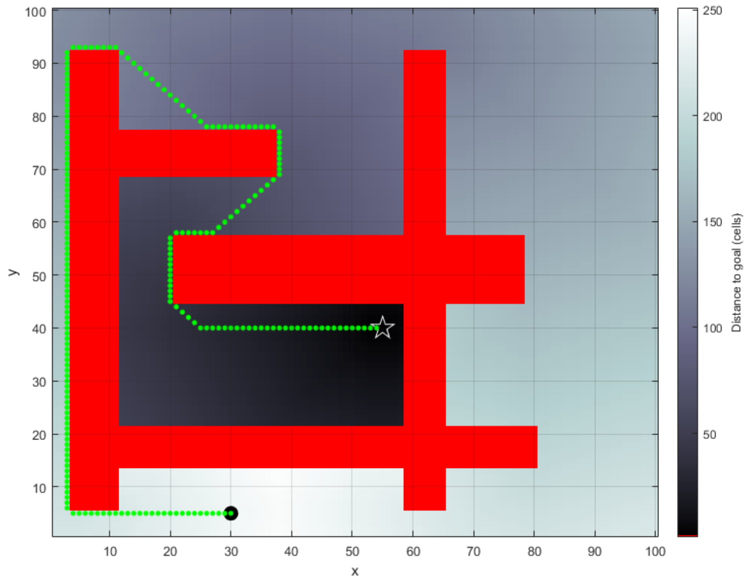


Figure 1: Plot of DXform.

The idea behind this path finding approach is that the robot knows its environment with a map and uses it to plan its path. The map calculates the distance to the goal of each cell and labels accordingly. The function **DXform** applies this **Distance Transform** algorithm to create the map which the robot uses in each iteration to go to the neighboring cell where the distance to the goal is minimal.

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

To create obstacles, I created a empty map which i can edit using **makemap** function. Since the "circle" creation throwed an error that I couldn't resolve, I created a rectangle instead. Next to it I created a polygon as well. Then, I saved the figure as bitmap and merged with the initial map.

```
map2 = makemap(100); % create obstacles
map = map + map2; % merge obstcles and initial map
dx = DXform(map);
dx.plan(goal);
dx.query(start, 'animate');
```

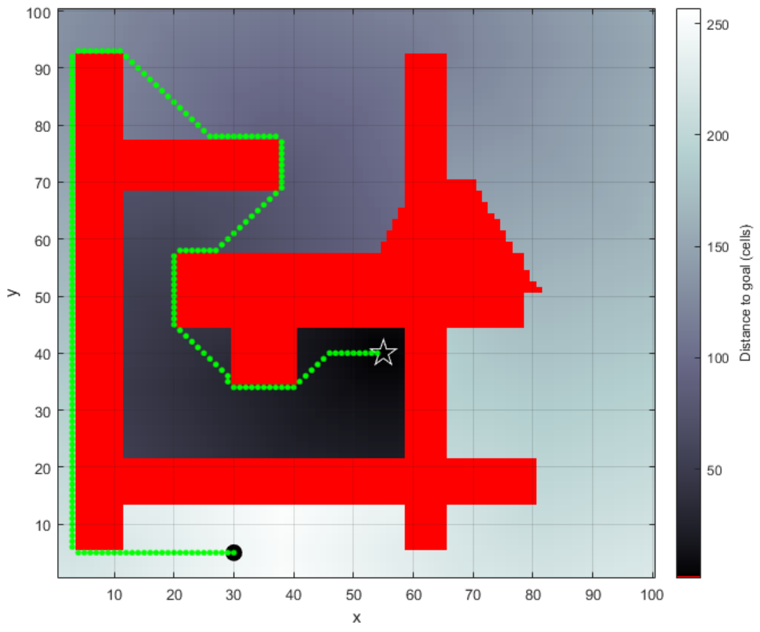


Figure 2: Plot of DXform with obstacles.

We see that the robot got around the rectangle obstacle as expected. Other than that there is not much a difference on the map, because the initial and goal positions are not changed. The problem with the Map Based Planning with the distance transform is that when the initial and goal positions are defined, its easy for robot to plan its path whether there is new obstacles or not but, when the initial or the goal position is changed, the map needs to be recalculated and the map labes need to change accordingly.

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* $12 \leq x \leq 50$ *and* $78 \leq y \leq 88$. *Discuss the resulting navigation of the robot, as compared with Problem 1. Justify your answer.*

For this problem, we need to use **Dstar** function. After examining the source code we found that to modify the costmap, there exists a **modify_cost** function. We created a simple nested for loops to modify the cost regarding the penalty area. Then, we need to run **plan** function to update the planned path with the new costmap.

```
load map1           % load map
goal = [55,40];
start=[30,5];
ds = Dstar(map); % create navigation object
ds.plan(goal)       % create plan for specified goal

% Create a new penalty box 12 <= x <= 50 && 78 <= y <= 88
for x = 12 : 50
    for y = 78 : 88
        ds.modify_cost([x; y], 2);
    end
end
ds.plan();

ds.query(start, 'animate')      % animate path from this start location
penalty_area = rectangle("Position",[12 78 50-12 88-78]);
penalty_area.EdgeColor = "yellow";
penalty_area.LineStyle = "--";
```

There is not much a difference in map labelling when compared with the Figure 1 at first look however, D* algorithm's method uses Graph Based Model, which calculates the cost function between the nodes where Distance Transform uses a Map Based Model. Graph Based Model's advantage is we can modify certain area's cost values. This can help to notion the non-ideal but passable areas in the map such as, inclined platforms, debrie fields which is hard to move on or slippery surface that can cause deviation more than expected. From the Figure 3 we can see the path crosses through the penalty area. This is because sometimes it is more cost efficient to pass through the penalty area shortly than getting around the penalty area.

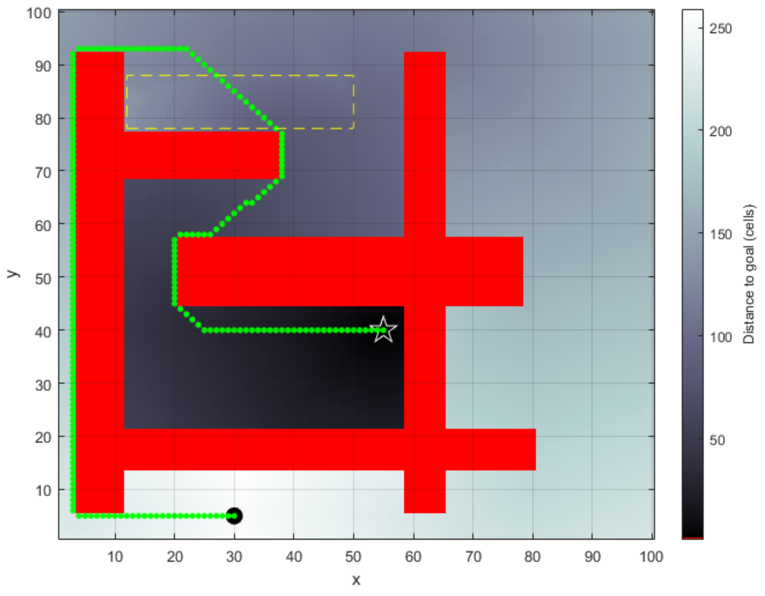


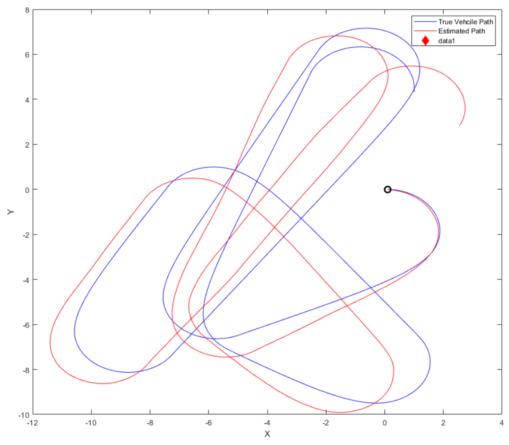
Figure 3: Plot of Dstar with a penalty area marked with yellow dashed lines.

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.*

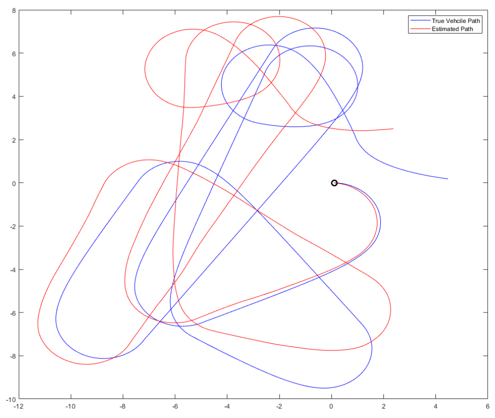
In this question, we aim to estimate a pose based on the previous one. Here is the code for this question:

```
% Define time steps
timesteps = [1000, 1200];

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

% Create a Bicyle model with noisy odemetry
veh = Bicycle('covar', V);

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

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

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

% Initial covarinace
P0 = diag([0.0005, 0.005, 0.001].^2);

% Apply Extended Kalaman Filter
```

```
ekf = EKF(veh, V, P0);

% Run the simulation for the timesteps
for i = 1 : length(timesteps)
    figure(7);
    ekf.run(timesteps(i));

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

    % Plot the uncertainty ellipses
    figure(i + 2);
    ekf.plot_ellipse();

    % Plot the covariance against time
    figure(i + 4);
    clf
    ekf.plot_P();
end
```

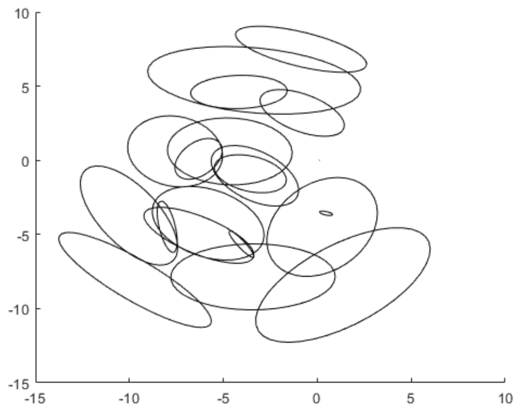


(a) 1000 timesteps.

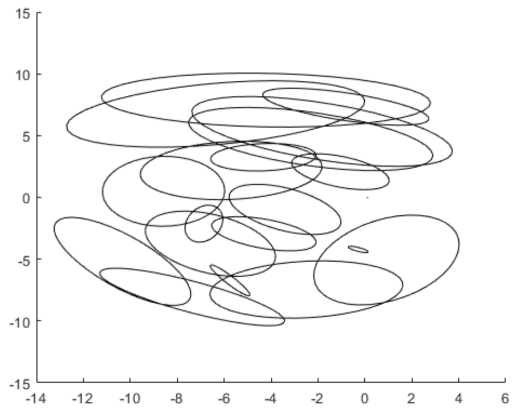


(b) 1200 timesteps.

Figure 4: Plot of true vs estimated path at 1000 & 1200 timesteps.

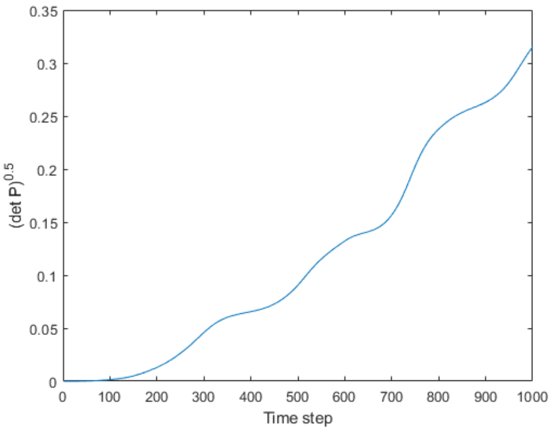


(a) 1000 timesteps.

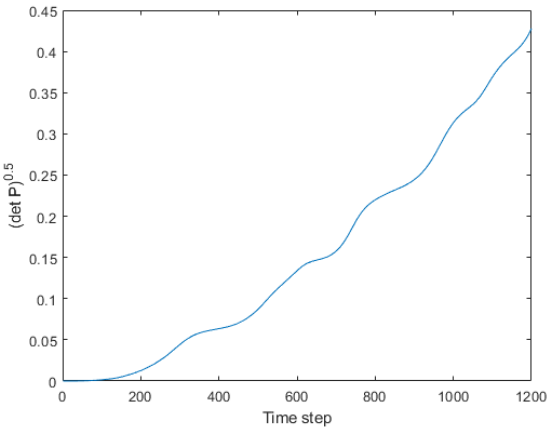


(b) 1200 timesteps.

Figure 5: Plot of uncertainty ellipses at 1000 & 1200 timesteps



(a) 1000 timesteps.



(b) 1200 timesteps.

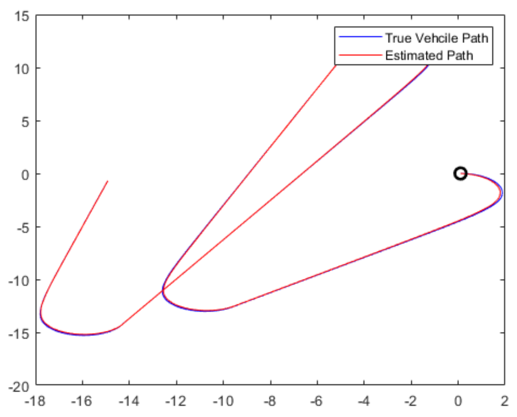
Figure 6: Plot of covariance at 1000 & 1200 timesteps.

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*

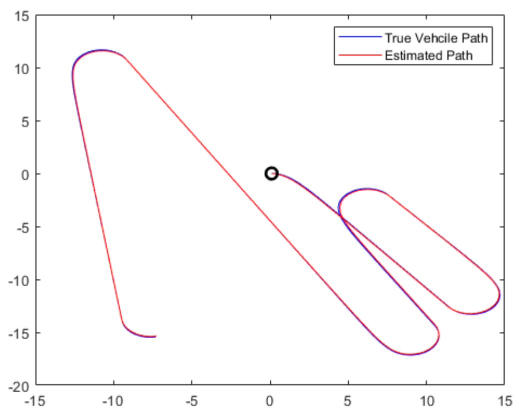
   Here is the code:

```
% Create odometry covariance per timestep
V = diag([0.005, 0.5 * pi / 180].^2);

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

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

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

```
% Create a sensor that uses the map and vehcile 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
% vehcile state covariance P0
ekf = EKF(veh, V, P0, sensor, W, map);

% Run the simulation for the timesteps
for i = 1 : length(timesteps)
    figure(i);
    ekf.run(timesteps(i));

% Plot the true vehcile path and the estimated path
    figure(i + 9);
    veh.plot_xy("b");
    hold on;
    ekf.plot_xy("r");
    legend("True Vehcile Path", "Estimated Path");
end
```



(a) 1000 timesteps.



(b) 1200 timesteps.

Figure 7: Outputs at 1000 & 1200 timesteps.

1 References

Corke, P. (2017). Robotics, vision and control fundamental algorithms in MATLAB. Springer International Publishing.

Cem Tolga Münyas, Boğaçhan Ali Düşgül