Mechanical Engineering Department
ME 654 Advanced Robotics
Spring 2016

Simulation Lab 2

Rapidly-Exploring Random Tree (RRT) planning algorithm in conjunction with the Kalman Filter (KF) for localization.

Deng Xiong

In this simulation-based assignment, I utilized MATLAB to implement a procedure for mobile robot path planning under localization uncertainty, which utilizes the Rapidly-Exploring Random Tree (RRT) planning algorithm in conjunction with the Kalman Filter (KF) for localization. The code package includes a main m. file, obstacle generator file, ellipse generator file and two collision check file which are based on teacher's script. In the main function file, it generates 100 RRT paths by iteration, determines the shortest one and predicts the maximum and minimum localization uncertainty, visualizing the localization uncertainty encountered along planned paths by ellipses. I was following the instruction on Canvas. In the instruction, the goal of the assignment is to implement a path planning procedure that will guide our robot to a desired goal region while both avoiding collisions and managing the growth of localization uncertainty. Below shows a two-step process:

1. Using the RRT algorithm to identify a collision-free path that reaches the desired goal region
2. Applying the KF to the resulting planned path to compute the anticipated localization uncertainty

This process will be iterated to construct a large set of feasible paths with different sensing and localization outcomes, allowing us to choose the path that offers a desirable trade-off between distance traveled and localization uncertainty. I used RRT pseudocode(Fig.1) to implement the algorithm in MATLAB. MATLAB code is attached in the package.

For a general configuration space C, the algorithm in pseudocode is as follows:

```
Algorithm BuildRRT
  Input: Initial configuration q_init, number of vertices in RRT K, incremental distance Δq)
  Output: RRT graph G

  G.init(q_init)
  for k = 1 to K
    q_rand ← RAND_CONF()
    q_near ← NEAREST_VERTEX(q_rand, G)
    q_new ← NEW_CONF(q_near, q_rand, Δq)
    G.add_vertex(q_new)
    G.add_edge(q_near, q_new)
  return G
```

- "←" is a shorthand for "changes to". For instance, "*largest* ← *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the value that follows.

In the algorithm above, "**RAND_CONF**" grabs a random configuration q_{rand} in C. This may be replaced with a function "**RAND_FREE_CONF**" that uses samples in C_{free}, while rejecting those in C_{obs} using some collision detection algorithm.

"**NEAREST_VERTEX**" is a straightforward function that runs through all vertices v in graph G, calculates the distance between q_{rand} and v using some measurement function thereby returning the nearest vertex.

"**NEW_CONF**" selects a new configuration q_{new} by moving an incremental distance Δq from q_{near} in the direction of q_{rand}. (According to [4] in holonomic problems, this should be omitted and q_{rand} used instead of q_{new}.)

Figure 1 RRT pseudocode

An RRT grows a tree rooted at the starting configuration by using random samples from the search space. As each sample is drawn, a connection is attempted between it and the nearest state in the tree. If the connection is feasible (passes entirely through free space and obeys any constraints), this results in the addition of the new state to the tree. The length of the connection between the tree and a new state is frequently limited by a growth factor. If the random sample is further from its nearest state in the tree than this limit allows, a new state at the maximum distance from the tree along the line to the random sample is used instead of the random sample itself. The random samples can then be viewed as controlling the direction of the tree growth while the growth factor determines its rate. This maintains the space-filling bias of the RRT while limiting the size of the incremental growth.

First, generate_obstacles.m was used first to map the obstacles, the start point and the goal region. A random point was then generated in the plane of 100^2 as a target point and collision_check_segment.m file was used to check if there is collision between the obstacles and the line from start point to target point. If there is no collision, a new point was generated by increasing 2m along the direction of the line. Besides, collision_check_point.m file was used to check if the new point has collision with the obstacles. If there is no collision, the new point was set as the initial point. This process is iterated until the initial point reaches the goal region. Then to take a sampled node and the RRT tree as input and return the index of the nearest neighbor in the tree and line the path point.

$$\mathrm{x}_{predicted} = \mathbf{A}\mathrm{x}_{n-1} + \mathbf{B}\mathbf{u}_n$$

$$\mathrm{P}_{predicted} = \mathbf{A}\mathrm{P}_{n-1}\mathbf{A}^{\mathbf{T}} + \mathbf{Q}$$

$$\tilde{\mathrm{y}} = \mathbf{z}_n - \mathbf{H}\mathrm{x}_{predicted}$$

$$\mathrm{S} = \mathbf{H}\mathrm{P}_{predicted}\mathbf{H}^{\mathbf{T}} + \mathbf{R}$$

$$\mathrm{K} = \mathrm{P}_{predicted}\mathbf{H}^{\mathbf{T}}\mathrm{S}^{-1}$$

$$\mathbf{x}_n = \mathrm{x}_{predicted} + \mathrm{K}\tilde{\mathrm{y}}$$

$$\mathbf{P}_n = (I - \mathrm{K}\mathbf{H})\mathrm{P}_{predicted}$$

Figure 2 Kalman filter equations

Un = Control vector. This indicates the magnitude of any control system's or user's control on the situation.
Zn = Measurement vector. This contains the real-world measurement we received in this time step.
Outputs:
Xn = Newest estimate of the current "true" state.
Pn = Newest estimate of the average error for each part of the state.
Constants:
A = State transition matrix. Basically, multiply state by this and add control factors, and you get a prediction of the state for the next time step.
B = Control matrix. This is used to define linear equations for any control factors.
H = Observation matrix. Multiply a state vector by H to translate it to a measurement vector.
Q = Estimated process error covariance. Finding precise values for Q and R are beyond the scope of this guide.
R = Estimated measurement error covariance. Finding precise values for Q and R are beyond the scope of this guide.

Kalman filter equation (Fig. 2) was programmed in the main m.file.Then KF was used to predict the localization uncertainty realized over the paths produced by RRTs. The KF is to predict uncertainty as part of a path planning procedure, so the filter will be used with some reduced functionality. We will be assuming, for the purpose of predicting uncertainty, that the robot achieves maximum likelihood measurements and actions. You may assume that at all times, the robot is successfully able to follow its planned path, and is not perturbed from it by process noise. You may also assume that at all times, the robot collects a perfect measurement, which is not corrupted by sensor noise. However, in spite of this we will assume that the robot’ s error covariance matrix P evolves according to the prescribed zero-mean Gaussian white process noise w, with covariance Q, and zero-mean Gaussian white senor noise v, with covariance R. These terms will contribute to the growth of anticipated localization uncertainty, even if they do not cause the state estimate x to diverge from the planned path.

100 instances of the path planning was implemented and 3 plots of (1) the shortest path obtained(Fig.3), (2) the path of minimum uncertainty at the terminal state(Fig.4), and (3) the path of maximum uncertainty at the terminal state across all trials(Fig.5) were generated. The shortest path was found in 51th iteration. I found that the size of the uncertainty depends on the length of the path. A longer paths usually attach with larger uncertainties. Besides, the robot determine its location by sensing the obstacles so uncertainty is usually small around the obstacles.

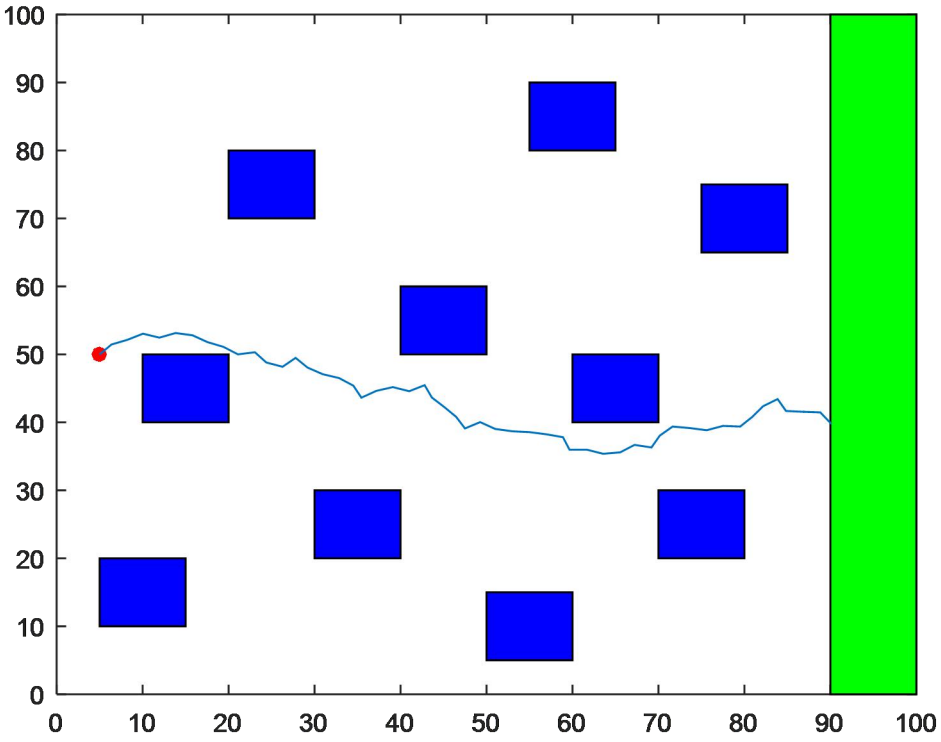

Figure 3 The shortest path

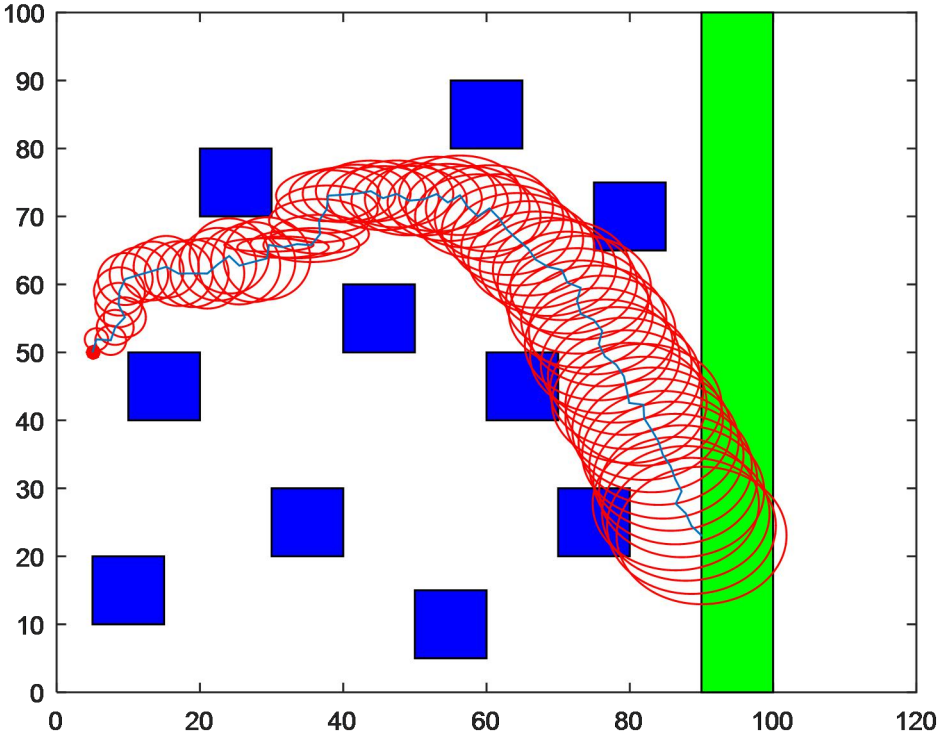

Figure 4 The path of maximum uncertainty at the terminal state

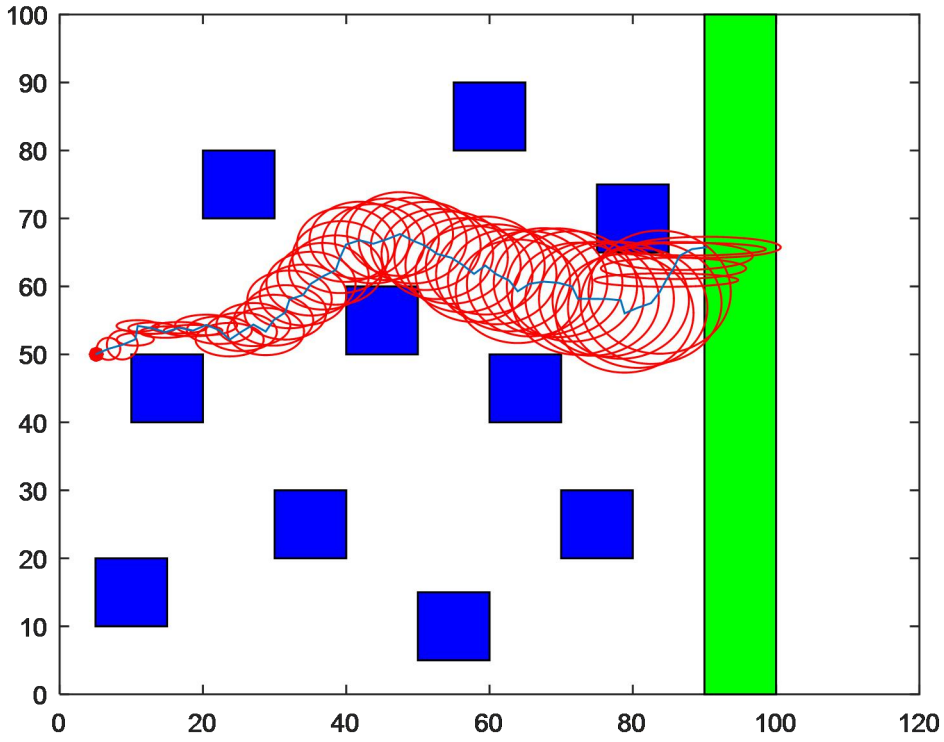

Figure 5 The path of minimum uncertainty at the terminal state,