The Interaction Between Mobile Robot and Manipulator

Runqiu Shi Zechen Wang Shervin Rahimimanesh Ziyi Wu

Mentor: Prof. Heather Culbertson

Abstract

This paper will explain our experimental results for the USC CSCI445 final project. The goal of the project is to realize a mobile robot that carries an object through a maze to a robotic arm, which picks up the object and places it on a shelf. We proposed particle filter, rapidly exploring random tree (RRT), odometry, PID control and inverse kinematics methods to achieve this goal.

Key Words: particle filter, RRT, odometry, PID, inverse kinematics, CSCI445

1 Motivation

The idea comes from how to improve factory autonomy in the trend of Industry 4.0. The main goal for this paper narrows this vision into the grounding task that robots transport goods and place them into specific locations without human intervention.

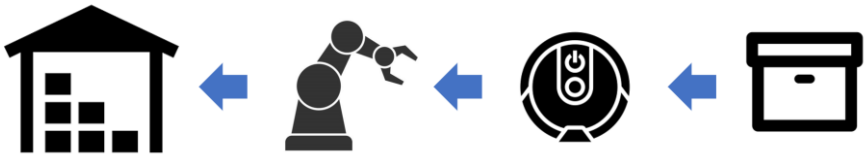


Fig.1 The Flowchart for Automatic Process

2 Methods

We propose techniques consisting of particle filter, RRT, odometry, PID and inverse kinematics to reach the goal. Besides, we creatively use ensemble methodology like bagging and boosting algorithms in machine learning to combine particle filter with odometry to realize robot localization.

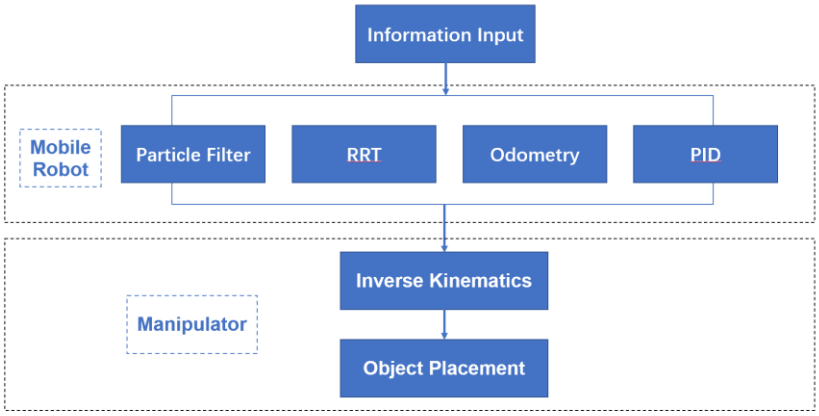


Fig.2 Workflow of the Algorithms

3.1 Particle Filter

Particle filter is a great extension model from traditional Hidden Markov Chain Model (HMM) like Kalman Filter. It is easy to adapt the particle filter for non-linear and non-Gaussian systems. That is the large benefit of the particle filter compared with Kalman Filter, which is the reason why we choose particle filter as a localization method. Besides, Particle filter is a Bayesian model by exploiting past state information and current measurement to get posterior probability. Then, by utilizing the resampling step, particle filter can get the most likely location of the current robot, dropping unlikely virtual robots in the map.

Algorithm 1 Particle filter

1. Initialization $t = 0$
   for $i = 1, \ldots, N$
   sample $X_0^{(i)} \sim p(X_0)$; $t := 1$
   endfor
2. Sampling step
   for $i = 1, \ldots, N$
   sample $\tilde{X}_t^{(i)} \sim P(X_t | X_{t-1}^{(i)})$
   $\tilde{X}_{0:t}^{(i)} = \left(X_{0:t-1}^{(i)}, \tilde{X}_t^{(i)}\right)$
   endfor
   for $i = 1, \ldots, N$
   evaluate importance weight $\tilde{w}_t^{(i)} = \left(Y_t, \tilde{X}_t^{(i)}\right)$
   endfor
   normalize the importance weight
3. Selection step
   for $i = 1, \ldots, N$
   draw N with probability proportional to $\tilde{w}_t^{(i)}$
   add $\tilde{X}_t^{(i)}$ to $X_t^{(i)}$
   endfor
   set $t \leftarrow t + 1$ and go to step 2

Fig.3 Particle Filter Pseudocode

3.2 Rapidly Exploring Random Tree (RRT)

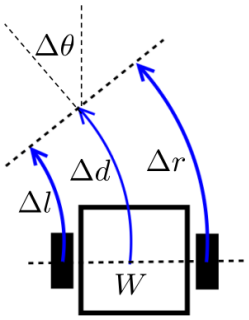
RRT is the path planning method that returns a reasonably good solution within a short time. RRT samples randomly from the configuration space and tries to construct a tree-like structure of paths. To extract the path, it simply searches backwards from the goal location to follow the tree structure. Since this map is small and configuration space is available, RRT seems to be a good choice.

```
GENERATE_RRT(x_init, K, Δt)
1  T.init(x_init);
2  for k = 1 to K do
3      x_rand ← RANDOM_STATE();
4      x_near ← NEAREST_NEIGHBOR(x_rand, T);
5      u ← SELECT_INPUT(x_rand, x_near);
6      x_new ← NEW_STATE(x_near, u, Δt);
7      T.add_vertex(x_new);
8      T.add_edge(x_near, x_new, u);
9  Return T
```

Fig.4 RRT Pseudocode

3.3 Odometry

We use odometry to calculate distance and orientation by counting encoders' readings. Since odometry is reliable and cheap, it is widely used in the automotive industry. Hence, we follow this selection to get the distance and orientation data.



$$\Delta d = \frac{\Delta r + \Delta l}{2}$$
$$\Delta\theta \approx \frac{\Delta r - \Delta l}{W}$$
$$x \leftarrow x + \Delta d \cos\theta$$
$$y \leftarrow y + \Delta d \sin\theta$$
$$\theta \leftarrow \theta + \Delta\theta$$

Fig.5 Illustration of Odometry

3.4 PID control

PID control is used in feedback control. It transforms the error into a commanded signal in actuators. By appropriate parameter tuning, the robot can reach a certain angle or location with great accuracy and low overshoot/damping.

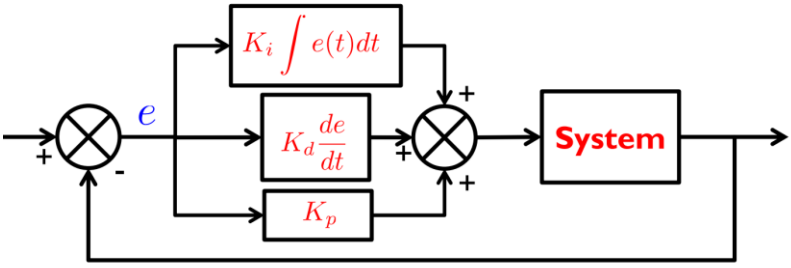


Fig.6 Illustration of PID control

3.5 Inverse Kinematics

Inverse Kinematics is a method for calculating the joint angles of a robotic arm given a desired position. We used inverse kinematics to control our robot arm to enable it to pick up the cup from the top of the robot and place it onto the shelf. The path of the robot arm was determined empirically to ensure that it would be able to move through space without dropping the cup or colliding with obstacles.

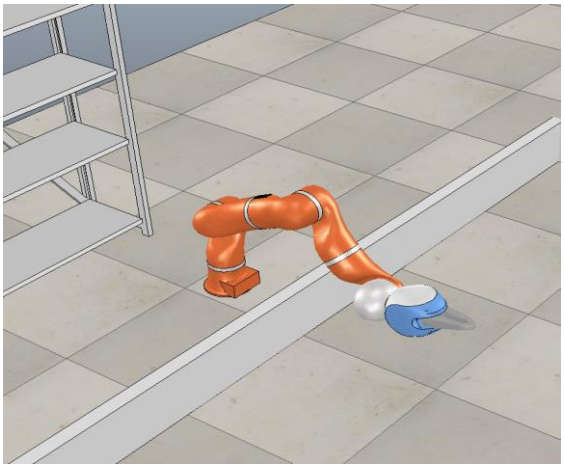


Fig.7 Robot at location (0.67, 0.2)

3.6 Algorithm Design

We combined the methods 3.1 - 3.4 on the robot so that it can plan, navigate and react at the same time. For the robotic manipulator, only method 3.5 is used. Our goal is to make the robot aware of its location and navigate its own way to the goal with obstacle avoidance, so that the manipulator can pick the load up in the end.

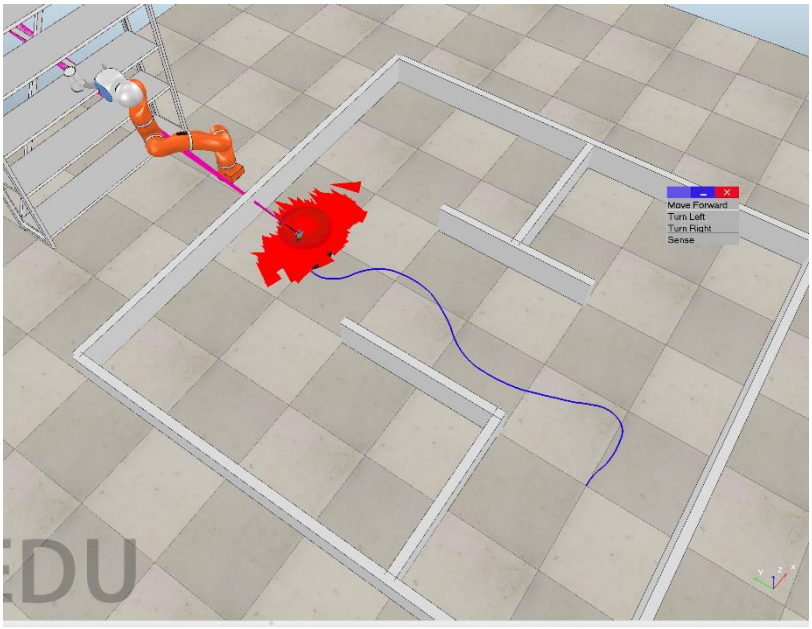


Fig.8 Illustration of Experimental Result

For the particle filter, since we have the knowledge of the starting point, we initialize the particle filter around our starting location. This method would largely increase the accuracy of the particle filter since the map itself is a nearly mirrored structure and sampling all over the map can be misleading. We decided to sense the environment and resample whenever a waypoint is reached, and in order to improve odometry data and to correct existing errors, we are updating the new odometry data as 0.9 * old odometry data + 0.1 * particle filter results each time when we resample.

Specifically, we used the following parameters for the particle filter:
num of particles: 500
translation variance: 0.05
rotation variance: 0.25
measurement variance: 0.1

For the RRT implementation, we take an input for the goal point, which has to be manually set in line 50 of finalproject.py, otherwise the default value of (1.5, 2.5) will be used. The program will be generating a path and visualizing it in the file project_rrt.png, which can be checked for reference.

Here are specific parameters:
num of trials: 800
Step size: 30

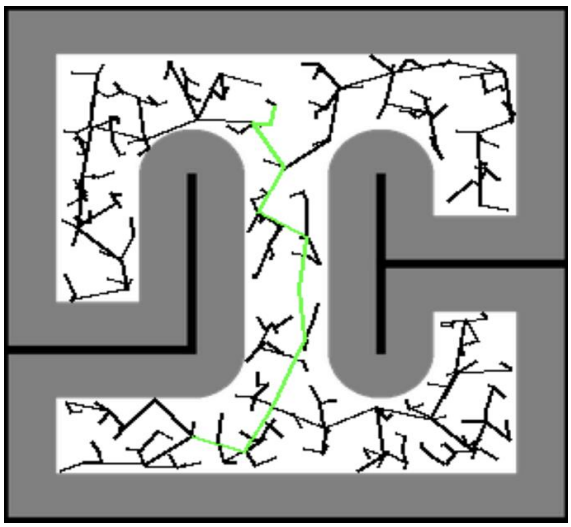


Fig.9 Illustration of RRT

In special cases where the RRT fails to find a path, try to increase num of trials and decrease the step size. It is guaranteed that the algorithm will eventually find a path if there exists one.

For Odometry and PID control, we implemented two modes for the robot to follow: go-to-goal and wall following. The robot frequently detects if there is an object in front of it and switch modes when necessary:

If no obstacles are detected, the robot will stay in go-to-goal until it reaches the final goal location. If there is an obstacle, the robot will automatically decide to turn left or right 90 degrees, and try to follow the wall at a specified distance using PID control. It will stay in the wall following mode for 2 seconds and call sonar detection again. If the path is now clear, it will enter a “skipping” mode to skip any waypoint that has already been passed during wall following. Besides, PID control parameters can be found at line 33 and 34 of finalproject.py

For Inverse kinematics, we implemented the inverse kinematics calculations in ik.py to determine the joint angles for joints 1 and 3 on the robot. For accessing x positions, we rotated the base of the arm and used the manipulator for fine movement controls. In order to slow down the speed of the arm to prevent the robot from dropping the cup, we made each motion occur over 10 steps.

4 Conclusion

We successfully realize the goal at the beginning of the paper after many trials. But, we can’t confirm whether TAs can get expected results when testing code, since there will always be a minor inaccuracy in the robot’s final destination in front of the manipulator, which can be critical as the cup is a very small and lightweight object.

There is another way we can optimize the accuracy of location. It is called Kalman Filter which sets Odometry as the main sensor measurement and uses sonar sensor results to correct Odometry by using Bayes Rule. Theoretically, the variance range will be smaller than adopting only one sensor. But this method is out of scope of this course and we will adopt it in the future.