Robot Localization and Navigation

Ziqi MA, *Student of SJTU*

***Abstract*—Nowadays, robot localization and navigation domain increasingly becomes a popular field, where navigating a robot autonomously is the most difficult problem and attract more and more people to pay attention. In this project, we aim to develop an algorithm for robot auto-navigation: In a certain given map, the robot need to plan a path and navigate from a start point to a destination point that are selected arbitrarily. The robot is equipped with some sensors that can get the inaccurate information of absolute position in the map and the range data about the environment. We realized the path-planning, state estimation and autonomous control of robot algorithm. We apply RRT algorithm to find an available path from starting point to destination, and then optimize the path by removing the path that difficult for robot to reach. Then, we use the measure data to roughly localize in the map and fine tune the robot state by ICP matching algorithm, after that, we precisely estimate robot state by EKF. Finally, we use preview control method to guide robot to the destination through the smooth path that we optimize. We evaluate our algorithm in the simulated environment. Our algorithm can find a path that suitable in the robot navigation problem and can rapidly navigate the robot to the destination without collide.**

***Index Terms*—localization and navigation autonomously, path planning, state estimation, robot control, RRT, ICP, EKF.**

I. Introduction

LOCALIZATION algorithms are part of our daily life and core for robotics. Recursive Bayesian estimation composes the current state-of-art in the field and has been essential for the development of localization, mapping, navigation and searching applications [1] [2]. Bayesian filters [3], e.g., Kalman and particle filters, are able to estimate the state of a system from noisy sensor observations formalized as a hidden Markov model. These approaches are useful also in the case of non-linear modeled systems and out-of-sequence measurements [4]. Kalman filter(KF) is an algorithm that uses a series of measurements observed over time, including statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each instance. [5] The particle filter (PF) is an approximate Bayesian method that tractably computes the posterior distribution of the state of any system given the observations. The state distribution is represented by individual particles, which are evaluated and weighted recursively. Particles with higher probability get bigger weights and are re-sampled into more particles in its neighborhood, whereas particles with smaller weights get fewer new samples close to them [6].

In this project, we need to solve a robot localization and navigation problem: In a certain given map, the robot need to plan a path and navigate autonomously from a start point to a destination point that are given arbitrarily. The robot is equipped with some sensors that can get the inaccurate information of absolute position in the map and the range data about the environment.

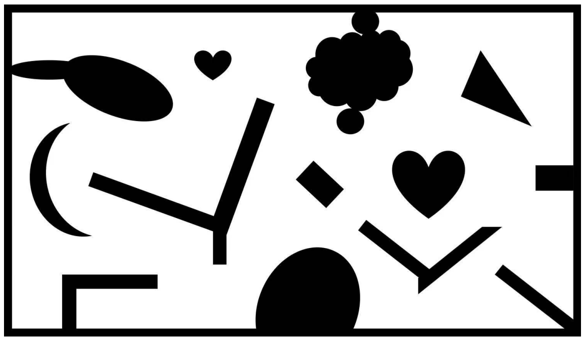


Fig. 1. Navigation map.

In order to find an available path from starting point to destination, we implement Rapidly-exploring Random Trees(RRT) algorithm and then we optimize the path by modifying the path that difficult for robot to reach. Then, we use the given data to roughly localize the robot and range data in the map and fine tune these data by Iterative Closest Point(ICP) matching algorithm, After that, we precisely estimate robot state by Extended Kalman Filter(EKF) with fine-tuned robot location and orientation and range data matching in map. Finally, we use preview control method to guide robot to the destination through the smooth path that we optimize.

The report is developed as follow: In Section II, we describe the problem which needs to be solved; In Section III, we present the methods that we used; In Section IV, we write how we implement the algorithms; In Section V, we show the performance of our auto-navigation algorithm; In Section VI we write our thought about the project and the improvement in the future and in Section VII, we conclude the whole project.

II. Problem Statement

A two-dimensional (2D) map is given and shown in 1. The robot is equipped with a 2D laser scanner that outputs range data about the environment i.e. the relative positions of its surrounding objects in the environment. The robot is also equipped with a Bei-Dou GPS that outputs absolute positioning measurements of the robot in the global reference. The robot needs to localize itself in the map based on range data, absolute positioning measurements, and certain a prior map info. Based on the localization result, the robot should be able to plan a path and navigate autonomously from an arbitrarily-given start point to an arbitrarily-given destination point.

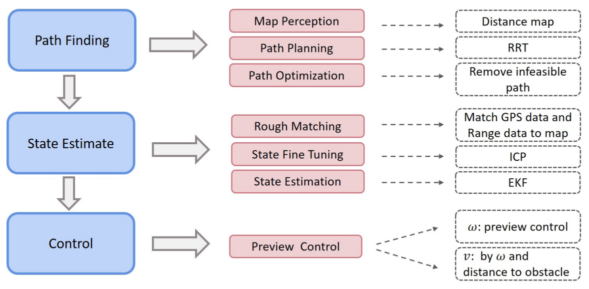


Fig. 2. Frame of robot navigation problem.

III. Method

The idea to navigate the robot to the destination is first find a path from starting point to destination, then localize the robot in the map which can be also regarded as robot state estimation and finally control the movement of robot. However, the absolute positioning of robot is very imprecise, through which is impossible to get robot's location directly, what's more, we don't know the information of robot initial direction, which causes the robot state estimation useless. On the other hand, range data has a small standard deviation and is relatively accurate. However, searching in the whole map and matching the range data to estimate the state of robot is quite time-consuming, especially when we don't know the direction of robot. So, the best way to do localization is to fuse these two kind of information to get a more precise robot position and direction and then do state estimation with EKF.

In this project, our work is based on the following assumptions: Firstly, the robot has the information of the whole map including the obstacles, the navigable place and the destination point, with which it can plan the path and do some operations based on the path. Secondly, the robot knows how range data is obtained and it can calculate the theoretical range data of a estimated state in the map. Thirdly, due to experience, the robot knows the bias and standard deviation of absolute positioning measurements and those of range data.

We separate the whole process in three parts: available path finding, robot state estimation and robot control, the whole frame of our work is shown in Fig.2.

A. Path Finding

In order to plan a path suitable for robot navigation, First of all, we create a distance map who notes the Manhattan distance of each pixel to obstacles, then we use a fast path planned algorithm, RRT algorithm to generate a path who doesn't collide with obstacle, in the end, we use path optimization algorithm to smooth the path.

1) Map Perception: We generate a distance map based on the two-dimensional map, which can be used in the following steps. The distance map has the same size as the map, however each pixel on the distance map notes the minimal distance to obstacles with Manhattan distance. The detail of generating distance map is shown in Algorithm1.

Algorithm 1 Generation of Distance Map

```
Initialize DMAP with a matrix of $inf$ as same size as MAP
DMAP$(\text{MAP} == 0) = 0$ // the distance of obstacle is 0
$W \leftarrow width(\text{DMAP})$
$H \leftarrow height(\text{DMAP})$
For $i = 2, ..., H$ do
    For $j = 2, ..., W$ do
        If $\text{DMAP}_{i,j} \neq 0$ then
            $\text{DMAP}_{i,j} = \min(\text{DMAP}_{i,j}, \text{DMAP}_{i-1,j}, \text{DMAP}_{i,j-1}) + 1$
For $i = i, ..., H-1$ do
    For $j = 1, ..., W-1$ do
        If $\text{DMAP}_{H-i,W-j} \neq 0$ then
            $\text{DMAP}_{H-i,W-j} = \min(\text{DMAP}_{H-i,W-j}, \text{DMAP}_{H-i+1,H-j}, \text{DMAP}_{H-i,W-j+1}) + 1$
```

2) Path Planning: Unlike most path planning algorithms, the main challenges imposed by navigation problem is that the robot does not have existing nodes to travel between. If we consider the famous Dijkstra's algorithm, that problem includes a graph. With a free continuous space, a graph of edges and vertices needs to be created. This fosters questions on the characteristics a graph should have for such a problem. There exist numerous path planning algorithms that address the navigation problem. Rapidly-exploring random trees(RRT) [7] is a common option that both creates a graph and finds a path. The premise of RRT is actually quite straight forward. Points are randomly generated and connected to the closest available node. Each time a vertex is created, a check must be made that the vertex lies outside of an obstacle. Furthermore, chaining the vertex to its closest neighbor must also avoid obstacles. The algorithm ends when a node is generated within the goal region, or a limit is hit. The pseudo code of RRT is shown in Algorithm 2.

Algorithm 2 Rapidly-exploring Random Trees (RRT) Code

```
QGOAL // region that identifies success
COUNTER = 0 // keeps track of iterations
LIM = N // number of iterations algorithm should run for
G(V,E) // Graph containing edges and vertices, initialized as empty
While COUNTER < N
    $X_{new}$ = RandomPosition()
    If IsInObstacle($X_{new}$) == True:
        continue
    $X_{nearest}$ = Nearest(G(V,E),$X_{new}$) // find nearest vertex
    $Link$ = Chain($X_{new}$,$X_{nearest}$)
    G.append($Link$)
    If $X_{new}$ in QGOAL:
        return G
return G
```

3) Path Optimization: However, the path found by RRT is just a path which does not collide with the obstacle, it doesn't consider the situation in robot navigation problem. The better path for robot to navigate cannot be closed to obstacle and cannot have lots of acute angle, as the former increase the

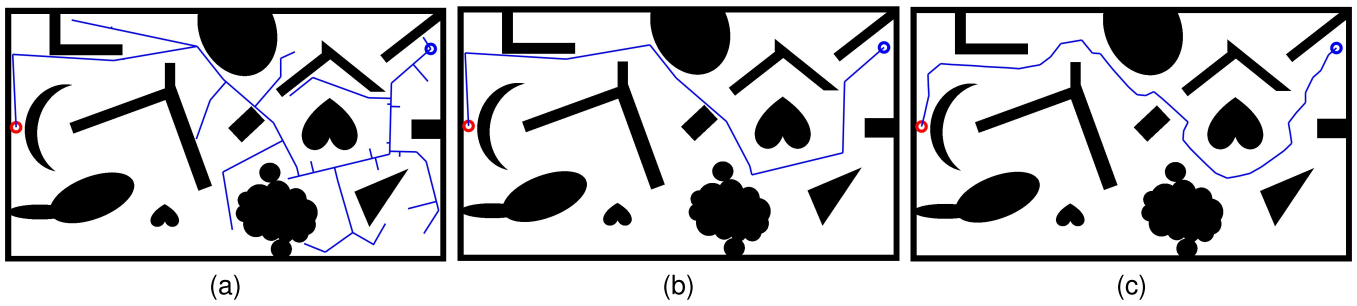


Fig. 3. Result of path finding algorithm. Blue circle is a starting point and red circle is a destination point. (a) Path planning by RRT algorithm. (b) Available path planned by RRT. (C) Final path that we optimize.

difficulty for state estimation and makes robot easy to collide, while the latter increases the difficulty for robot control.

Considering this situation, in order to plan a path suitable for robot control, we do the following changes. Firstly, we make the planned path far away from obstacles, we divide the whole chain of RRT path into small segments, so achieving local target one by one leads the robot to achieve the final goal, and then we use the distance map to optimize the local target point as far away from obstacle as possible in a constraint area. These points can be represent by:

$$G = \{(x_1^G, y_1^G), (x_2^G, y_2^G), ..., (x_n^G, y_n^G)\}$$

The first point in G represents the start point and the last point in G represents the destination point. After that, we calculate the angles corresponding to the target points, that is the angle of g_i in triangle $\triangle g_{i-1}g_ig_{i+1}$, where $g_i = (x_i^G, y_i^G) \in G$. If the angle calculated is smaller than the threshold angle, we remove the target point from the set G, until there is no angle surpass the threshold. In this project, we set the threshold as $\frac{11}{18}\pi$. In the end, we check again the final set G, if the segment former by neighbour target is still closed to the obstacle, we abandon this set G and re-operate the Algorithm 2 and former process, until a suitable target ensemble is found.

B. Robot State Estimation

The data that we have is absolute positioning measurements obtained by Bei-Dou GPS data and the relative positions of its surrounding objects in the environment obtained by 2D laser, we roughly localize the robot and range data in the certain prior map. Then, we use the ICP algorithm to fine tune the range data matched in environment and the location and orientation of robot. In the end, we fuse the information of fine-tuned range data and robot state by EKF to precisely estimate robot state.

1) Rough Matching: position given by GPS only have two dimensions (x_t, y_t), which omits the orientation data, however, this kind of information can be inferred by range data $LD = \{(xR_1, yR_1), (xR_2, yR_2), ..., (xR_n, yR_n)\}$, as it detects the relative positions of its surrounding objects in the environment. if we match the environment data with certain prior map successfully, we can get the localization and orientation of robot precisely.

We sample the possible robot state $\mathbf{z_{pt}} = (zx_t, zy_t, z\theta_t)$ in an area around GPS position (x_t, y_t) with the initial orientation of robot between $[-\pi, \pi]$. For each sampled state, we can recalculate range data in the environment $LE = \{(xR_1^w, yR_1^w), (xR_2^w, yR_2^w), ..., (xR_n^w, yR_n^w)\}$,

$$\begin{bmatrix} xR_i^w \\ yR_i^w \end{bmatrix} = \begin{bmatrix} cos(z\theta_t) & -sin(z\theta_t) \\ sin(z\theta_t) & cos(z\theta_t) \end{bmatrix} \begin{bmatrix} xR_i \\ yR_i \end{bmatrix} + \begin{bmatrix} zx_t \\ zy_t \end{bmatrix}$$

If the range data match to the environment, the distance of each point must be closed to the obstacle, so we find the rough robot state $\mathbf{z_{rp}} = (zr_x, zr_y, zr_\theta)$ who can get the minimal distance d in distance map:

$$\min_{zx_t, zy_t, z\theta_t} d(\mathbf{z_{pt}})$$
$$s.t. \quad d(\mathbf{z_{pt}}) = \textstyle\sum_{i \in LE} \text{DMAP}\left(\begin{bmatrix} xR_i^w \\ yR_i^w \end{bmatrix}\right)$$

2) State Fine Tuning: After having obtained the rough robot state, and rough range data in environment by matching, we can then calculate the theoretical range data with rough matching position and orientation(It is the second assumption), then the two dimensional Iterative Closest Point (ICP) algorithm [8] [9] is applying to range data and theoretical range data to adjust the orientation and position of robot slightly. If we note the theoretical range data as A and the rough range data as B, the main idea of ICP is that to find a rotation matrix R and a translation matrix T that minimize the distance after transition:

$$D(R, T) = \frac{1}{n} \sum_{i=1}^{n} \|A_i - (RB_i + T)\|^2$$

where n is the number of points in range data. After ICP process, we measure the robot state $\mathbf{z_t} = (z_x, z_y, z_\theta)$ more precise where:

$$z_x = zr_x + T(x)$$
$$z_y = zr_y + T(y)$$
$$z_\theta = zr_\theta + h(R)$$

where h is the function to transit rotation matrix to radian. And the matched range data in environment $zR = \{(xR_1^w, yR_1^w), (xR_2^w, yR_2^w)...(xR_n^w, yR_n^w)\}$

$$zR = R \times LE + T$$

3) Extended Kalman Filter (EKF): The Extended Kalman Filter (EKF) is the nonlinear version of the Kalman filter which linearizes about an estimate of the current mean and covariance. The system model is written as:

$$\mathbf{x}_t = g(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mathbf{e}_t$$

and measurement model is written as:

$$\mathbf{z}_t = h(\mathbf{x}_t) + \gamma_t$$

In the robot navigation problem, the arguments used in system model are position of robot and orientation of robot, with little increment, we can linearize system model as:

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} \approx \begin{bmatrix} \bar{x}_t \\ \bar{y}_t \\ \bar{\theta}_t \end{bmatrix} + \mathbf{A}(\mathbf{x_{t-1}}, \mathbf{u_t}) \begin{bmatrix} \Delta x_t \\ \Delta y_t \\ \Delta\theta_t \end{bmatrix} + \mathbf{B}(\mathbf{x_{t-1}}, \mathbf{u_t}) \begin{bmatrix} \Delta v_t \\ \Delta\omega_t \end{bmatrix}$$

where $\bar{x}_t, \bar{y}_t, \bar{\theta}_t$ are the value predict from last instance,

$$\begin{bmatrix} \bar{x}_t \\ \bar{y}_t \\ \bar{\theta}_t \end{bmatrix} \approx \begin{bmatrix} x_{t-1} + v_t\Delta T\cos(\theta_{t-1} + \omega_t\Delta T/2) \\ y_{t-1} + v_t\Delta T\sin(\theta_{t-1} + \omega_t\Delta T/2) \\ \theta_{t-1} + \omega_t\Delta T \end{bmatrix}$$

$\mathbf{A}(\mathbf{x_{t-1}}, \mathbf{u_t})$ represents the derivative of g with respect to $\mathbf{x}$, while $\mathbf{B}(\mathbf{x_{t-1}}, \mathbf{u_t})$ represents the derivative of g with respect to $\mathbf{u}$:

$$\mathbf{A}(\mathbf{x_{t-1}}, \mathbf{u_t}) = \begin{bmatrix} 1 & 0 & -v_t\Delta T\sin\left(\theta_{t-1} + \omega_t\Delta T/2\right) \\ 0 & 1 & v_t\Delta T\cos(\theta_{t-1} + \omega_t\Delta T/2) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{B}(\mathbf{x_{t-1}}, \mathbf{u_t}) =$$

$$\begin{bmatrix} \Delta T\cos\left(\theta_{t-1} + \omega_t\Delta T/2\right) & -v_t\Delta T^2\sin\left(\theta_{t-1} + \omega_t\Delta T/2\right)/2 \\ \Delta T\sin\left(\theta_{t-1} + \omega_t\Delta T/2\right) & v_t\Delta T^2\cos\left(\theta_{t-1} + \omega_t\Delta T/2\right)/2 \\ 0 & \Delta T \end{bmatrix}$$

For measurement model, we measure the distance of the robot to its range data:

$$z_t \approx \bar{z}_t + \mathbf{H}(\mathbf{x_t}) \begin{bmatrix} \Delta x_t \\ \Delta y_t \\ \Delta\theta_t \end{bmatrix} + \gamma_t$$

where $\bar{z}_t$ represents the euclidean distance from robot to range data:

$$\bar{z}_t = \sqrt{(x_t - xR_i^w)^2 + (y_t - yR_i^w)^2}$$

and $\mathbf{H}(\mathbf{x_t})$ is the partial derivative with respect to each argument:

$$\mathbf{H}(\mathbf{x_t}) = \begin{bmatrix} (x_t - xR_i^w)/\bar{z}_t & (y_t - yR_i^w)/\bar{z}_t & 0 \end{bmatrix}$$

So the process of EKF can be written as in Algorithm 3.

Algorithm 3 Extended Kalman Filter (EKF) Code

```
**System model:** $\mathbf{x}_t = g(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mathbf{e}_t$
**Measurement model:** $\mathbf{z}_t = h(\mathbf{x}_t) + \gamma_t$
**Prediction:**
$\overline{\mathbf{x}}_t = \begin{bmatrix} \bar{x}_t \\ \bar{y}_t \\ \bar{\theta}_t \end{bmatrix} = \begin{bmatrix} \hat{x}_{t-1} + \hat{v}_t\Delta T\cos\left(\hat{\theta}_{t-1} + \hat{w}_t\Delta T/2\right) \\ \hat{y}_{t-1} + \hat{v}_t\Delta T\sin\left(\hat{\theta}_{t-1} + \hat{w}_t\Delta T/2\right) \\ \hat{\theta}_{t-1} + \hat{w}_t\Delta T \end{bmatrix}$
$\bar{\Sigma}_t = \mathbf{A}\left(\hat{\mathbf{x}}_{t-1}, \hat{\mathbf{u}}_t\right)\hat{\Sigma}_{t-1}\mathbf{A}\left(\hat{\mathbf{x}}_{t-1}, \hat{\mathbf{u}}_t\right)^T$
    $+\mathbf{B}\left(\hat{\mathbf{x}}_{t-1}, \hat{\mathbf{u}}_t\right)\begin{bmatrix} \Sigma_v & 0 \\ 0 & \Sigma_w \end{bmatrix}\mathbf{B}\left(\hat{\mathbf{x}}_{t-1}, \hat{\mathbf{u}}_t\right)^T$
**Update:**
  **For** $i$ in range data:
      $\bar{z}_t = h\left(\overline{\mathbf{x}}_t\right) = \sqrt{\left(\bar{x}_t - xR_i^w\right)^2 + \left(\bar{y}_t - yR_i^w\right)^2}$
      $\mathbf{H}\left(\overline{\mathbf{x}}_t\right) = \begin{bmatrix} \left(\bar{x}_t - xR_i^w\right)/\bar{z}_t & \left(\bar{y}_t - yR_i^w\right)/\bar{z}_t & 0 \end{bmatrix}$
      $\mathbf{K} = \bar{\Sigma}_t\mathbf{H}\left(\overline{\mathbf{x}}_t\right)^T\left(\mathbf{H}\left(\overline{\mathbf{x}}_t\right)\bar{\Sigma}_t\mathbf{H}\left(\overline{\mathbf{x}}_t\right)^T + \Sigma_\gamma\right)^{-1}$
      $\hat{\mathbf{x}}_t = \overline{\mathbf{x}}_t + \mathbf{K}\left(z_t - \bar{z}_t\right)$
      $\hat{\Sigma}_t = \left(\mathbf{I} - \mathbf{K}\mathbf{H}\left(\overline{\mathbf{x}}_t\right)\right)\bar{\Sigma}_t$
```

C. Robot Control

We consider the situation of driving a car along a winding road. It is impossible for the driver to know complete information on the status of the road from the starting point to the destination. If the driver looks at a short range just before their eyes, then they can only steer the handle late after they recognize a curve. This will make the car deviate from the lane, and will lead to a car crash. In order to drive safely, the driver usually vies to look ahead as far as possible, and steer the handle properly based on the obtained future information on the lane. This means that the driver is unconsciously doing the control with previewed reference signal, since the lane can be treated as a reference signal or a reference trajectory which should be tracked by the car. This observation will be helpful for us to navigate a robot automatically with a planned path.

In this problem, we use preview control as the method of control. We remark that for movement of a robot, the argument that we control is the speed of robot position evolution v_t, and the speed of robot orientation evolution ω_t at one instance,

$$\mathbf{u_t} = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix}$$

after state estimation, the estimated value of robot state at instance t is,

$$\hat{\mathbf{x}}_t = \begin{bmatrix} \hat{x}_t \\ \hat{y}_t \\ \hat{\theta}_t \end{bmatrix}$$

with the path information G, we can calculate the point $(x_i^G, y_i^G) \in G$ that the robot aims to reach at instance t, so we calculate ω_t as:

$$\omega_t = \arctan\left(\begin{bmatrix} \cos(-\hat{\theta}_t) & -\sin(\hat{\theta}_t) \\ \sin(\hat{\theta}_t) & \cos(-\hat{\theta}_t) \end{bmatrix}\begin{bmatrix} x_i^G - \hat{x}_t \\ y_i^G - \hat{y}_t \end{bmatrix}\right)$$

When talking about the control input v_t, it depends on the distance between robot and obstacles and control input ω_t. If the robot is near the obstacle, we give v_t a small value to

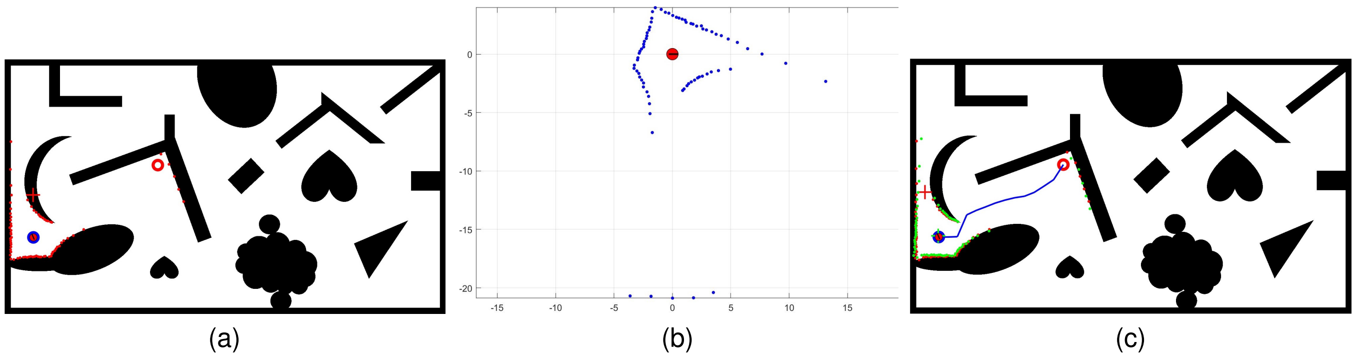


Fig. 4. Simulation Environment and Range Data for the robot. On the navigation map in left, red hollow circle and blue hollow circle represent the start point and destination point respectively, robot is abstracted as a red circle of diameter 1 with a black line indicating the direction faced and the small red points in the map is rang data.On the navigation map in right, planned path is blue lines and we add the estimated robot state by green cross and the matched range data in environment by small green point.

avoid collide in the following, while the robot is far away from obstacle, v_t is given a relatively larger value. Similarly, if the control input ω_t is large, we give v_t a small value to circumvent large evolution of state in one instance, if the control input ω_t is small, which means the robot is as close as possible to the path that we plan, we can give a larger input v_t to navigate faster.

IV. Implementation

We remark that the process of ICP to fine tune the rough robot state and rough range data is quite time-consuming, i we apply this in every control period, this will waste lot of time. In the experiment of simulation, we observe that when the initial state of robot is well estimated, the robot state is easy to estimate precisely because the robot move slowly during one timestamps. Considering of this characteristic in robot navigation, we use ICP algorithm only in the initial state estimation. Furthermore, we found that the range data that robot perceived is also too enough for state estimation, so in order to save time, we use only $\frac{1}{4}$ range data to do the process of state estimation including rough match and EKF, in the following periods.

The simulation environment (two-dimensional map) is shown in Fig. 4a, where black area represents obstacles and white area represents navigable space. Users can pick the start point and destination successively which are represented by blue and red hollow circle respectively. Robot is abstracted as a red circle of diameter 1 with a black line indicating the direction faced. Besides, range data is shown by some small red points in the map however the information about range data that we can get is shown in Fig. 4b, and absolute positioning measurement is represented by a red cross.

In order to better display the logic of robot navigation, we add some symbols in the figure, as shown in 4c the estimated state of robot is represented by a green cross. The planned path is indicated by blue lines, and the matched range data in environment is shown by small green points in the map. Except for the main file *RobotNavigation.m*, the fonctionality of each file in the folder is shown as follow:

- PathPlanning folder for path planning
  1) *DistanceMap.m*: to generate the distance map.
  2) *PlanningRRT.m*: to get the RRT results with the start point and destination point.
  3) *FindAvailablePath.m*: to find the path from start point to destination point with RRT results.
  4) *PathSmoothing.m*: to divide the planned path into small target and adjust slightly these target to avoid collide.
  5) *TakeoutAcute.m*: to take out the angle corresponding to small target point that surpass the threshold.
  6) *IsBadPath.m*: to verify if the final planned path is closed to obstacles.
- Matching folder for state estimation
  1) *ICPMatching.m*: to find the matrix R and T by ICP.
  2) *Matching.m*: to do the initial Location process.
  3) *MatchingAfterfirst.m*: to adjust the location and orientation of robot
  4) *RobotSensorInMap.m*: to calculate the theoretical range data with rough robot state.
- Control folder for robot control
  1) *PointCompare.m*: to find the next point in the local target set that robot need to reach.
  2) *CalculateOmega.m*: to calculate the control input ω.
  3) *CalculateV.m*: to calculate the control input v.
- ProfFunc folder: files given by professor
  1) *IsCollision.m*: to verify if collision happened
  2) *RobotAPS.m*: to get the robot absolute positioning measurement.
  3) *RobotDynamics.m*: to go to the next state of robot with the control input.
  4) *RobotRanging.m*: to get the range data.
- Visualization folder
  1) *DisplayRobot.m*: Given by professor to display true robot state.
  2) *PlotEdges.m*: to plot the the edge between two point.

V. Results

As shown in Fig.3, after we have chosen a start point and a destination, the algorithm begin to apply RRT algorithm (Fig.

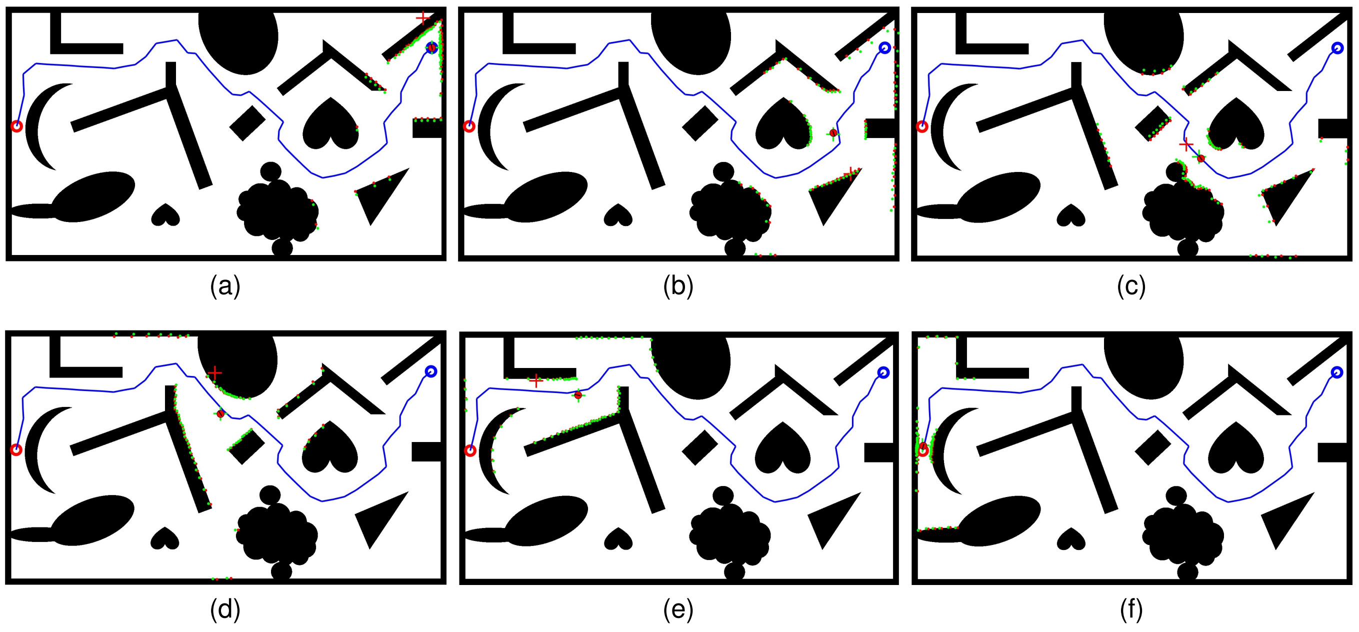

(a) (b) (c) (d) (e) (f)

Fig. 5. Robot navigation process. We observe that the robot can navigate itself automatically through the smooth path that we planned

3a), then we find an available path planned by RRT (Fig. 3b), and we optimize the path to avoid collide as possible (Fig. 3c).

Then we launch the robot, the process is shown in Fig. 5. red circle is the true state of the robot which we cannot obtain directly. however, by using the GPS information (red cross in the Fig. 4a) and range data (shown in Fig. 4b) We estimate the precise state of the robot, which is drawn as green cross (shown in Fig. 4c), and range data is also matched to the ground truth. We observe from Fig. 5 that the robot can navigate itself automatically through the smooth path that we optimized.

The error of EKF is shown in Fig.6. We notice that state estimated by using EKF is very closed to the true state compared to GPS positioning measurement, this verified what we learned on the class.

VI. Discussion

The effect of path optimization is significant, from Fig.3b, the path of RRT algorithm is somewhere close to obstacle and easy for robot to collide however, if we better optimize the path like Fig. 3c we can avoid this problem. However, our method that optimize the path may add extra distance on the path, this will add time for robot navigation, although comparing the situation of collision, it is easy to accept the extra distance adding to the path. In the future, we need to continuously optimize our path smooth algorithm to find an optimal path for robot navigation.

In order to ensure the accuracy, we do the state estimation in every control loop, although we eliminate the most time consuming algorithm ICP matching in the following steps, however, the state estimation process also needs lots of time and is unnecessary. However, if the robot doesn't do state estimation even during one second, the robot state obtained by predict model may have a large deviation from the true state, this may cause catastrophe. In the future, we will try to remove the state estimation in some the steps while guaranteeing the accuracy of the algorithm.

VII. Conclusion

In this project, we have implemented autonomous navigation in the simulated environment. We apply RRT algorithm to find an available path from starting point to destination, and then optimize the path by removing the path that difficult for robot to reach. Then, we use the measure data to roughly localize in the map and fine tune the robot state by ICP algorithm, after that, we precisely estimate robot state by EKF. Finally, we use preview control method to guide robot to the destination through the smooth path that we optimize.

In the simulation, we observe that our algorithm can find a path that suitable for robot to navigate and the robot can navigate itself successfully and rapidly to the destination through the planned path.

References

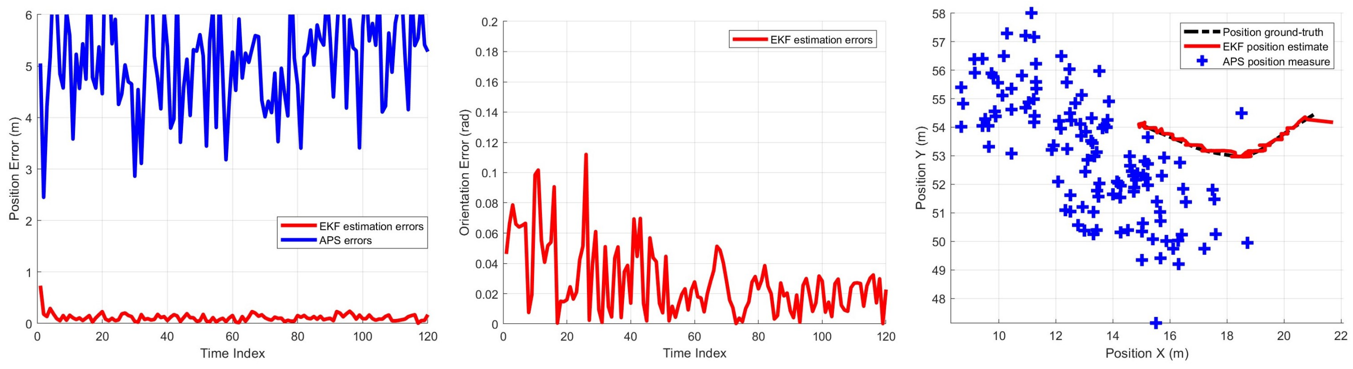
[1] P. Lanillos, E. Besada-Portas, J. A. Lopez-Orozco, and J. M. De la Cruz, "Minimum time search in uncertain dynamic domains with complex sensorial platforms," *Sensors*, vol. 14, no. 8, pp. 14 131–14 179, 2014. [Online]. Available: https://www.mdpi.com/1424-8220/14/8/14131

[2] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[3] V. Fox, J. Hightower, L. Liao, D. Schulz, and G. Borriello, "Bayesian filtering for location estimation," *IEEE pervasive computing*, vol. 2, no. 3, pp. 24–33, 2003.

[4] E. Besada-Portas, J. A. Lopez-Orozco, P. Lanillos, and J. M. De la Cruz, "Localization of non-linearly modeled autonomous mobile robots using out-of-sequence measurements," *Sensors*, vol. 12, no. 3, pp. 2487–2518, 2012.

[5] Y. Kim and H. Bang, "Introduction to kalman filter and its applications," in *Introduction and Implementations of the Kalman Filter*, F. Govaers, Ed. Rijeka: IntechOpen, 2018, ch. 2. [Online]. Available: https://doi.org/10.5772/intechopen.80600



(a) (b) (c)

Fig. 6. Errors of position and orientation. Red color represents EKF, blue color represents GPS positioning measurement and black color represents the ground truth of robot state. (a) Position errors of EKF and GPS data. (b) Orientation errors estimated by EKF. (c) Comparison of GPS data and robot state estimation with ground truth.

[6] B. Liu, S. Cheng, and Y. Shi, “Particle filter optimization: A brief introduction,” in *International Conference on Swarm Intelligence*. Springer, 2016, pp. 95–104.

[7] S. M. LaValle *et al.*, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[8] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. Spie, 1992, pp. 586–606.

[9] A. Censi, “An icp variant using a point-to-line metric,” in *2008 IEEE International Conference on Robotics and Automation*. Ieee, 2008, pp. 19–25.