Search-Based Planning and Reinforcement Learning for Autonomous Systems and Robotics

Than D. Le [1, 2]

[1] University of Bordeaux, France; than.ld@ieee.org
[2] Ton Duc Thang University;

Abstract: In this chapter, we address the competent Autonomous Vehicles should have the ability to analyze the structure and unstructured environments and then to localize itself relative to surrounding things, where GPS, RFID or other similar means cannot give enough information about the location. Reliable SLAM is the most basic prerequisite for any further artificial intelligent tasks of an autonomous mobile robots. The goal of this paper is to simulate a SLAM process on the advanced software development. The model represents the system itself, whereas the simulation represents the operation of the system over time. And the software architecture will help us to focus our work to realize our wish with least trivial work. It is an open-source meta-operating system, which provides us tremendous tools for robotics related problems.

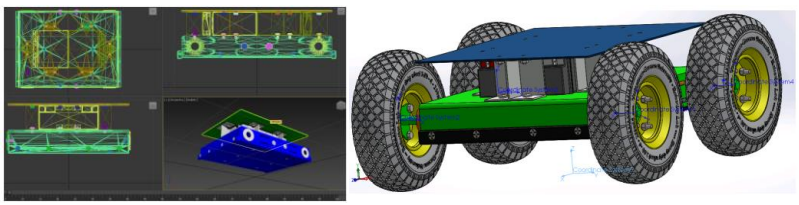
Specifically, we address the advanced vehicles should have the ability to analyze the structured and unstructured environment based on solving the search-based planning and then we move to discuss interested in reinforcement learning-based model to optimal trajectory in order to apply to autonomous systems.

Keywords: SLAM, Probabilistics Robotics, Kalman Filter, Extended Kalman Filter, Modelling, Simulation, Search-based Planning, Reinforcement Learning, Monte Corlo, Q-Learning;

---

1. Introduction

In past work, Deep learning [19] and (Deep) reinforcement learning [16], [22] included DeepMind and Deep Q-Learning mechanism [18] in 2014, AlphaGo won the champion in the game of Go [21] in 2016 and also OpenAI and PPO occurred 2017 [20] are currently resolutions in applied artificial intelligence for vehicle and autonomous systems. Basically, deep learning is the best model for data representation and reinforcement learning is a modern approach to solving the decision-making. Before moving forward to modern approaches. there are essential to represent the basic things to making the intelligent autonomous systems that can be enabled to solve basic level based on simultaneous localization and mapping (SLAM) based on interacting the unknown environment.
Currently, the state-of-the-art in research-based planning and replanning [17] for autonomous and mobile systems. Moreover, motion planning [1], [3], [8] will be explored the probabilistic problems capturing from the Figure 1.1 and Figure 1.2



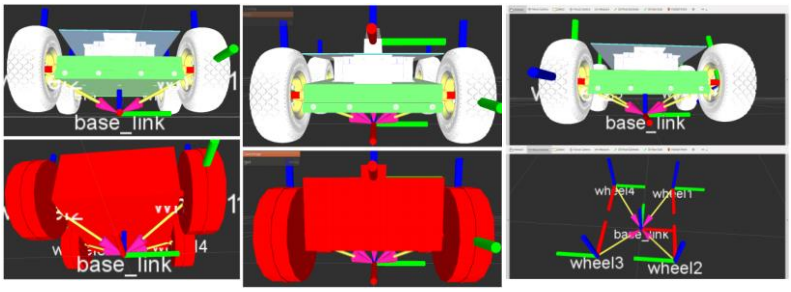


Figure 1.1: Modelling and Simulation: A sample for Vehicle Mobility

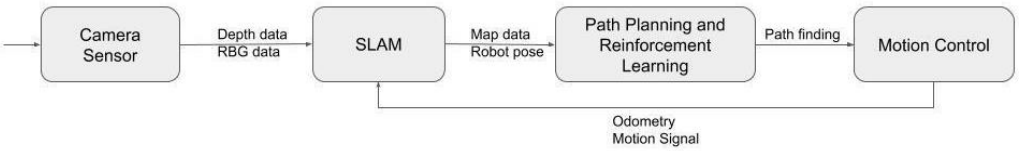


Figure 1.2: System Architecture of Search-Based Planning and Reinforcement Learning in Advanced Vehicle Mobility

On paper, the next section will be described the path planning based on representing the search-based planning to find the shortest path in heuristical representation. Next section, it will be covered the uncertainty problems by using the SLAM concepts. Then next, we will introduce the reinforcement learning for generating the trajectory path in maze environment. Finally, it concluded with some experiments in both simulations and real-world.

2. Path Planning

In this section, we will explore the path finding algorithms are to study compares some popular algorithms: Dijkstra's Algorithm, Best-First-Search and The A* Algorithm. Initially, the controlling functions are written in both Python and C++ language. Through doing this, we learned more about the language, being able to write a publisher and subscriber nodes, understand how operating systems works like ROS as an open source software, how to develop software without reinventing the wheel. At the end, we successfully write a navigation app that implements A* Shortest Path Finding Algorithms to make the Vehicle or autonomous robots walking to the 2D path generated by software developments. The source code are well-commented, the packages are well-organized and are uploaded to bitbucket, making it easier for other contributors to continue the project for universities or organizer. Firstly, we can define the theory concepts below.

2.1. Dijkstra's Algorithm and Best-First-Search

A common example of a graph-based path finding algorithm is Dijkstra's algorithm such as Figure 2.1. Dijkstra's algorithm is a shortest path finding algorithm conceived by computer scientist Edsger W. Dijkstra in 1956. It works by visiting a set of open nodes in the graph starting with the starting node. It then repeatedly examines the closest node with the lowest distance cost that have not been examined, adding it to the set of closed node (nodes that have been examined). It expands outwards from the starting point until it reaches the goal. If there are no negative edge node (node with the negative distance cost), Dijkstra's algorithm is guaranteed to find a shortest path from the starting point to the goal, since the lowest distance nodes are examined first. In the following map, the star is

the starting point, the "X" is the goal, the white path is the calculated path and the blue and area inside it is the areas Dijkstra's algorithm have scanned.

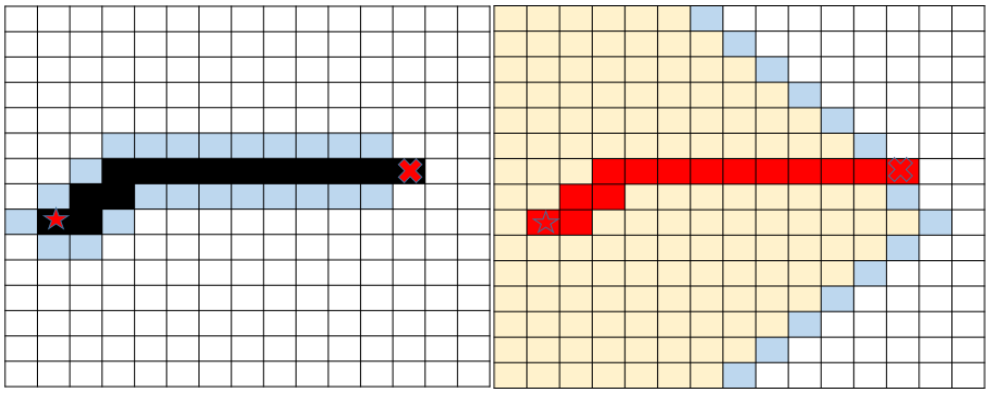


Figure 2.1: LEFT: Path finding using Dijkstra's algorithm; RIGHT: Greedy Best-first search algorithm

The Greedy Best-first search is a search algorithm which explores a map of nodes by expanding the most promising node chosen based on Evaluation Function $f(n)$. Different from Dijkstra's algorithm, which selects the node closest to the starting point, Greedy Best-first search algorithms selects the node closest to the goal. By using the priority queue ordering, Greedy best-first search algorithms tends to focus on paths that lead directly to the goal.

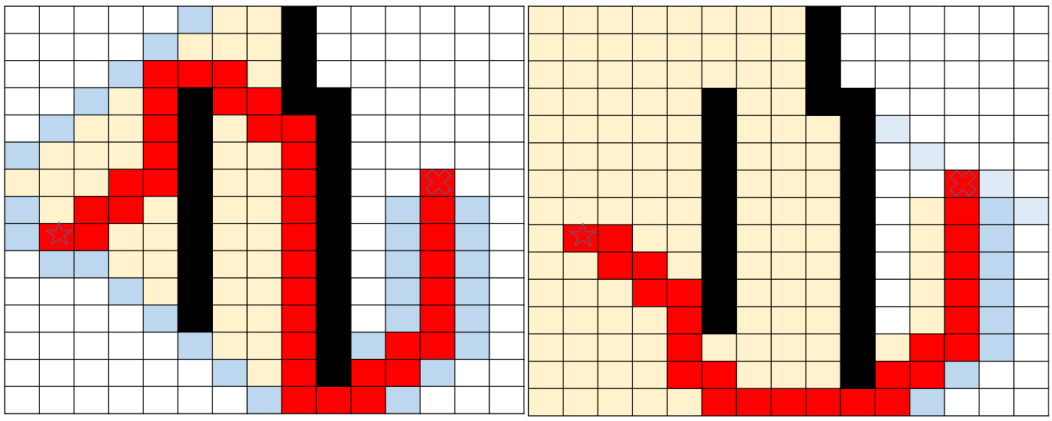


Figure 2.2: Path finding using

Amazing, according to the pictures, Dijkstra's algorithm is outclassed completely by Greedy Best-first search algorithms. However, Greedy Best-First Search is not guaranteed to find a shortest path. Let's see what happens in a more complex map.

Dijkstra's algorithm still have to scan a large amount of nodes, but it found the shortest path:

On the other hand, Greedy Best-First-Search algorithm works much more faster Figure **[LEFT], but the path it generated is much longer:

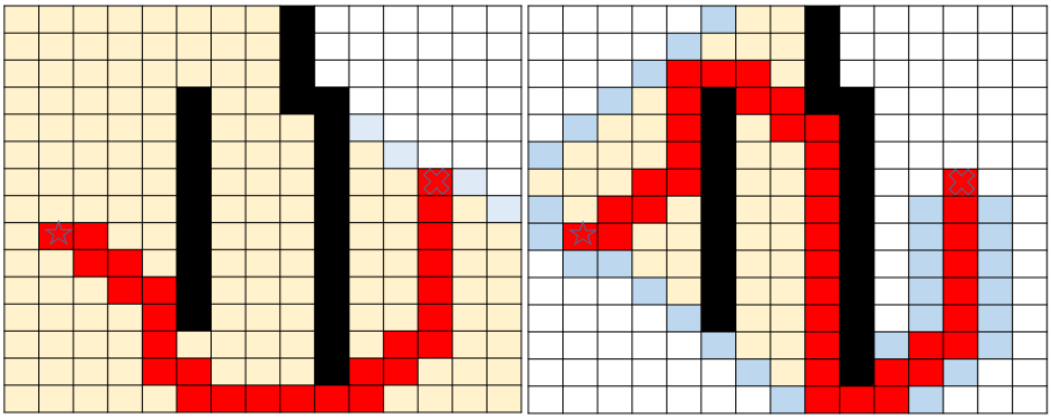


Figure 2.3: Path Finding Problem using the Dsitristra

The problem is that Greedy Best-First-Search is "greedy" and it keeps trying to move towards the goal even if it's a longer path, and Dijkstra's algorithm takes too much time and resource to find the path. So can we combine the best of both algorithms? In order to solve this question, In 1968, AI researcher Nils Nilsson developed a new path finding Algorithms - A* (A Star Algorithms). The algorithms combines Greedy Best-First-Search and Dijkstra's algorithm. Although A* is built on top of the heuristic, and the heuristic itself does not give you a guarantee, A* can guarantee a shortest path

2.1. A Shortest Path Finding Algorithm*

A* Shortest Path Finding Algorithm Peter Hart was first described by Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International) in 1968. It is an extension of Edsger Dijkstra's 1959 algorithm. Since then, it has become the leading pathfinding algorithm. A* Algorithm is widely used in map navigation and graph traversal, the process of plotting an efficiently traversable path between multiple nodes.

A* is a best-first search algorithms, meaning that it will choose the path considered as the best solution (least distance traveler, shortest time, etc.) by searching among all possible paths to the target.

As we have mentioned before, Dijkstra's Algorithm is accuracy to find the shortest path, but it wastes time exploring in directions that aren't promising while Greedy Best First Search explores in promising directions but it may return the longer path result. The A* algorithm calculates both the actual distance from the start and the estimated distance to the goal so it can guarantee to find shortest path while taking much less time Dijkstra's Algorithm.

First, let's define the cost function:

$$f(n) = g(n) + h(h) \qquad (1)$$

With f is the cost of a node. The lower the f, the better the node. The g is the cost it took to get to the node, for example: the length of the road which we passed by from the start. h is the heuristic score function, the tentative cost to reach the goal from that node. The accuracy of your path depends on how "good" is your h. Very rarely (if ever) is your h perfect.

Enough for theory, let's see what A* Algorithms method can do.

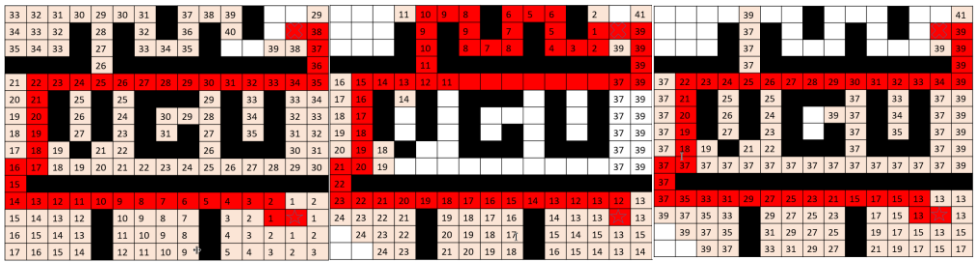


Figure: Comparison of Path finding using 3 different algorithms: Left - Dijkstra's Algorithm; Center - Greedy Best-First Search; Right - A* Search

From the figure above Figure , we can clearly see that A* algorithms finds a path as good as what Dijkstra's algorithm found while took less time to scan the node. (scan a smaller number of nodes).

Let's consider how algorithms work. First, we create an open list and closed list. The open node list start with the start node and contains all nodes that have not yet been checked. The closed node list stores all node that have been visited.(moved from the open node list). The algorithm works by maintaining these two lists. The core loop of the algorithm selects a node from the open list with the lowest estimated cost (f) to reach the goal. If one of the selected node is the goal, the search will be stopped. Else, it calculate the node cost then push all the valid direction nodes (8 nodes around the current node) into the open list. Then the checked node is moved to the closed node list.The process repeats until the path is generated.

Algorithm 1: AStar($s_{\text{start}}, s_{\text{goal}}$)

Input : start node s_{start}, goal node s_{goal}
Output: shortest path P from s_{start} to s_{goal}.

begin
  $G = \emptyset$; /* G-score look-up table */
  $T = \emptyset$; /* Parent reference list */
  $F = \emptyset$; /* F-score priority queue */
  $G[s_{start}] = 0$;
  $T[s_{start}] = NULL$;
  $F[s_{start}] = h(s_{start}, s_{goal}) + G[s_{start}]$;
  $s = s_{start}$;
  while $F \neq \emptyset$ *or* $s \neq s_{goal}$ **do**
    $s = F.ExtractMin()$;
    for *node* n *in* $Neighbor(s)$ **do**
      $g_{new} = G[u] + cost(s, n)$;
      if $n \notin T$ *or* $g_{new} < G[n]$ **then**
        $T[n] = s$;
        $G[n] = g_{new}$;
        $F[n] = h(n, s_{goal}) + G[n]$;
  $P = \emptyset$;
  $s = s_{goal}$;
  $P.append(s)$;
  while $s \neq s_{start}$ **do**
    $s = T[s]$;
    $P.append(s)$;
  $P.reverse()$;
  return P;

3. Uncertainty Representation using Kalman Filter Landmark Based SLAM

This chapter provides us the first insight of SLAM. The Kalman filter and theory of SLAM topic will be quickly covered before a more practical explanation at the last section of the chapter can tell basic things. The aim of the chapter is to introduce Kalman filter to the reader for ease of further SLAM understanding in the future.

3.1. Principle of Kalman filter

In the 1950s, Rudolf Emil Kalman published the famous article about the recursive approach to the discrete data linear filtering. The article was "A new Approach to Linear Filtering and Prediction Problems". From that time to now, with the dramatic development of digital calculation, Kalman filter has become a popular research topic, and has been applied in various areas, such as automatic electronics, robotics, radar technology, and so on.

The Kalman filter was originally designed to collect and incorporate data from sensors in adaptive way. Once system of equations and sensors statistic data is known and deterministic, the filter will give optimal estimation after filtering errors and noises. There are many ways to optimize the data based on some particular criteria, but the Kalman filter seem to be the best in linear Gaussian estimation when it can incorporate all data given to it.

In short, Kalman filter is a mathematical system of equations which describe a method for effectively recursive estimation of states such that the average of all variances and covariances of state variables becomes minimized. Kalman filter is very optimal for estimation of states in the past, the present, and the future, even if the accuracy of robotic models is not guaranteed. Kalman filter is of our interest under the assumption that the system is linear and the noise is Gaussian. There are two phase in the filter: *prediction* and *update*. At first, Kalman filter predicts an n dimensional predecessor state vector denoted by x'_t at time step t through the stochastic differential equation,

$$x'_t = A_t x_{t-1} + B_t u_t + w_t \quad (38)$$

, where A_t is the state transition matrix at time step t, B is the optional control input matrix at time step t that can affect x_t , and $w_t \sim N(0, R_t)$ indicates the additive white Gaussian noise (AWGN) of the transition. Next, the filter calculate the predecessor covariance matrix P'_t as follows,

$$P'_t = A_t P_{t-1} A_t^T + R_t \quad (39)$$

, where R_t is the variance matrix of the transition noise w_t at time step t. Then, the filter enters the second phase called *correction* or *update*, where the posterior state vector x_t and the posterior covariance matrix P_t are improved from the correspondent predecessor ones by the expressions below,

$$x_t = x'_t + K_t(z_t - H_t x'_t) \quad (40)$$

$$P_t = (I - K_t H_t) P'_t \quad (41)$$

, where H_t is the Jacobian matrix from measurement model $z_t = H_t x'_t + v_t$ at time step t
$v_t \sim N(0, Q_t)$ is the random measurement noise variable (AWGN), Q_t is the measurement noise variance matrix, I is the identity matrix, and K t is called *Kalman gain* at time step t,
and computed by,

$$K_t = P'_t H_t^T (H_t P'_t H_t^T + Q_t)^{-1} \quad (42)$$

If, at time step t, the observation error covariance R t decreases, the actual observation z_t is treated to be better than the prediction $H_t x'_t$. On the other hand, a smaller prior error covariance P'_t leads to a less important actual measurement z_t and weights the predecessor prediction x'_t more. The prediction phase and the correction phase of the Kalman filter are performed sequentially.
To summarize, at every time step t, the belief $p(x_t \vee u_t, x_{t-1})$ of the robot is represented by the mean μ_t and the covariance P_t, and the basic Kalman filter algorithm is:

(a) $\mu'_t = A_t \mu_{t-1} + B_t u_t$
(b) $P'_t = A_t P_{t-1} A_t^T + R_t$
(c) $K_t = P'_t H_t^T (H_t P'_t H_t^T + Q_t)^{-1}$
(d) $\mu_t = \mu'_t + K_t(z_t - H_t \mu'_t)$
(e) $P_t = (I - K_t H_t) P'_t$
(f) Obtain $bel(x_t)$as normal distribution expression with mean μ_t and variance P_t

The mathematical derivation as well as other knowledge on Kalman filter can be found at [5, 6, 7]

3.2. General landmark based SLAM

The SLAM problem can be various in categories and algorithms, but , in regards with landmark based method, the SLAM problem typically consists of following parts: landmark extraction, data

association, state estimation, state update and landmark update. Again, for each part, there are many ways to solve. This section is written according to [9], which gives a very potentially practical approach to the SLAM process.

In general speaking, landmarks are features which can easily be re-observed and distinguished from the environment. These are used by the robot to find out where it is as the same way human does.

As well as state variable, landmarks cannot directly sensed but we have to reliably extract them from the robot measurement data. There are two popular landmark extraction algorithms when most of robots uses a laser scanner.
Firstly, the spike landmark extraction uses extreme changes to find landmarks. They are designed to find values in the range of a laser scan where the distances to two values differ by more than a certain amount, for example, 0.5 meters.
Secondly, the random sampling consensus (RANSAC) is a method to extract lines from a laser scan. Once this is done, RANSAC checks how many laser readings lie close to this best fit line. If the number is above some threshold which is manually set by us, the robot can believe that it have seen a line. The code implementing RANSAC can be found at [11].

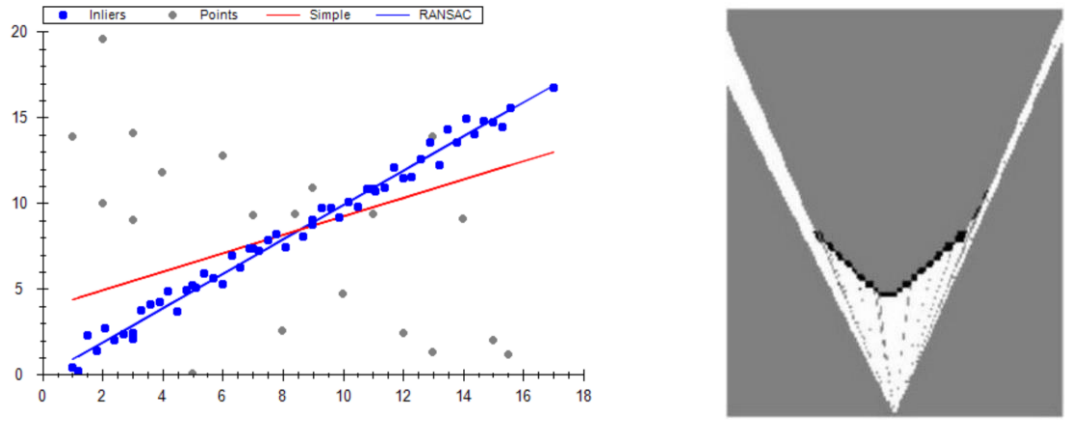


Figure 3: (**a**) (**b**)

The problem of data association is that of matching observed landmarks from different (laser) scans with each other. If the robot fails to re-observe and match landmarks, there are many problems like getting lost, building a wrong map, confused navigation, and so on. People have defined a data-association policy that deals with these issues. We assume that a database is set up to store landmarks the robot has seen. The database is usually initially empty. Then, there is a rule that not until the robot has seen a landmark N times (manually setting), the robot don't actually consider the landmark worthwhile to be used.
This will prevent the case where we extract a bad landmark, which is not easy to distinguish, re-observe and match again. This technique is called the nearest-neighbor approach as it rely on correlation between the nearest landmarks in the database.
With the landmark based, we need to improve some concepts from the previous probabilistic framework in previous session. We define m_i to describe a vector that represents the location of the i^{th} landmark whose true location is assumed time invariant. Also, the set of all landmarks will be denoted as

$$m = m_1, m_2, \ldots, m_n \quad (43)$$

The SLAM algorithm is now implemented in a standard two-step recursive (sequential) prediction and correction phases:

Prediction

$$p(x_t, m \vee x_0, z_{0:t-1}, u_{0:t}) = \int \quad p(x_t \vee x_{t-1}, u_t) p(x_{t-1}, m \vee z_{0:t-1}, u_{0:t-1}) dx_t - 1$$

Measurement update

$$p(x_t, m \vee x_0, z_{0:t}, u_{0:t}) = \frac{p(z_t \vee x_t, m) p(x_t, m \vee z_{0:t-1}, u_{0:t}, x_0 \ )}{p(z_t \vee z_{0:t-1}, u_{0:t})}$$

Unfortunately, most of environment where the robot is situated are chaotic, such that system of models are non-linear and non-Gaussian as well. In this case, the systems need to be linearized first before we can use the Kalman filter, which is the reason why the extended Kalman filter (EKF) appears.

As soon as the landmark extraction and the data association is ready, the SLAM process can be considered as three steps, which we have already known from previous chapters,

1. Predict the current state estimate using the odometric data
2. Update the predicted state from re-observing landmarks
3. Add new landmarks to the current state

More detail solution can be read over at [3, 12-15].

3.3. From the view of simulation

Now, we have a set of general ideas that SLAM may work. There are some simulation of our interest topic: EKF Landmark based SLAM, which was done by Jai. All related material can be found at [16].

The Figure 4a shows us a 2D projection map, where shapes with black solid line represents obstacles. Small blue crosses represents landmarks, and for some reasons, we assumes these landmarks can be re-observed through obstacles. It's a fact that radio frequency identification (RFID) landmarks can be used to make this possible. The thin line with circle markers is the path that robot will take, which represent for control actions given to the robot at every time step.

We can have a look at the Figure 4b when the simulation starts. At every same time step, the red triangle describes the robot with regards to it concurrently building map, and the blue triangle points out the true shapes and locations that the robot is taking in the real-life map. Here, at first, the robot seems to be consistent of where it is, so the blue triangle and the red triangle are about to overlap each other. The set of red crosses represents for the point clouds resulted from sensor data extraction, which can be used to build a map. Overall, the robot's pose and the landmarks' position have correspondent small red surrounded random ellipses, which indicate randomized Gaussian noises for two dimensional random variable of the coordinate.

In the Figure 4c, when the robot has made a significant SLAM, the state vector x and other parameter of Kalman filter are enlarged together with the data of new landmarks. At this point, the weak computer, where the simulation may run on, can cause more uncertainty, because of the problem between calculation time and real time requirement. Because of more uncertainty, which means the

robot lacks of information in the right time to compute exactly everything, the map become confused as in the Figure.4d. In the Figure 4c, the red triangle, which is the robot's location inside its mind, becomes a little far from its true position, the blue triangle. Also, everything inside its current maps is shifted in a similar way as well. This is very good thing, cause it tell us the strong correlation between landmarks and landmarks, landmarks and point clouds, point clouds and point clouds, which is the solution for the robot to correct from localization and mapping failure due to odometry data and real time problem.

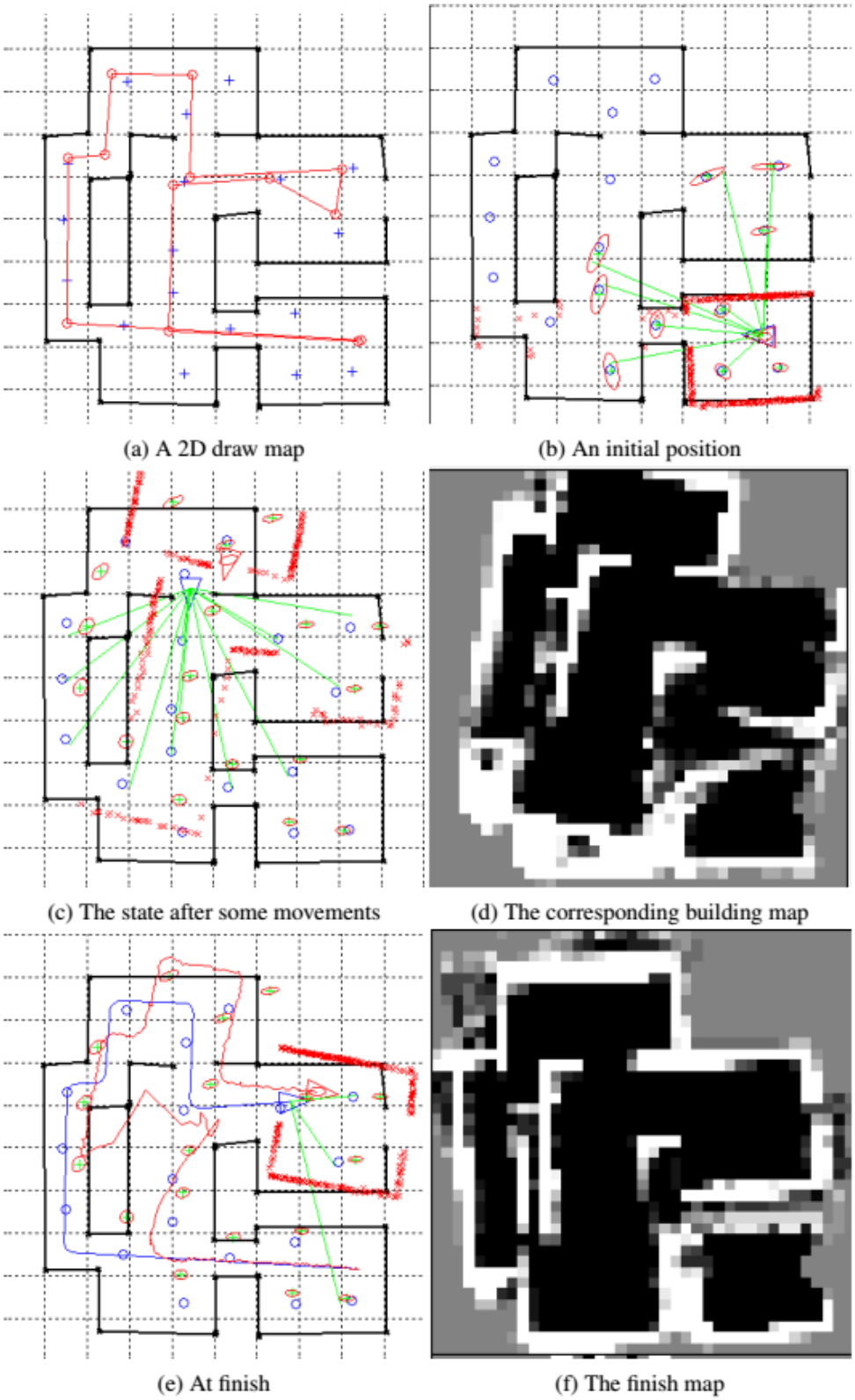
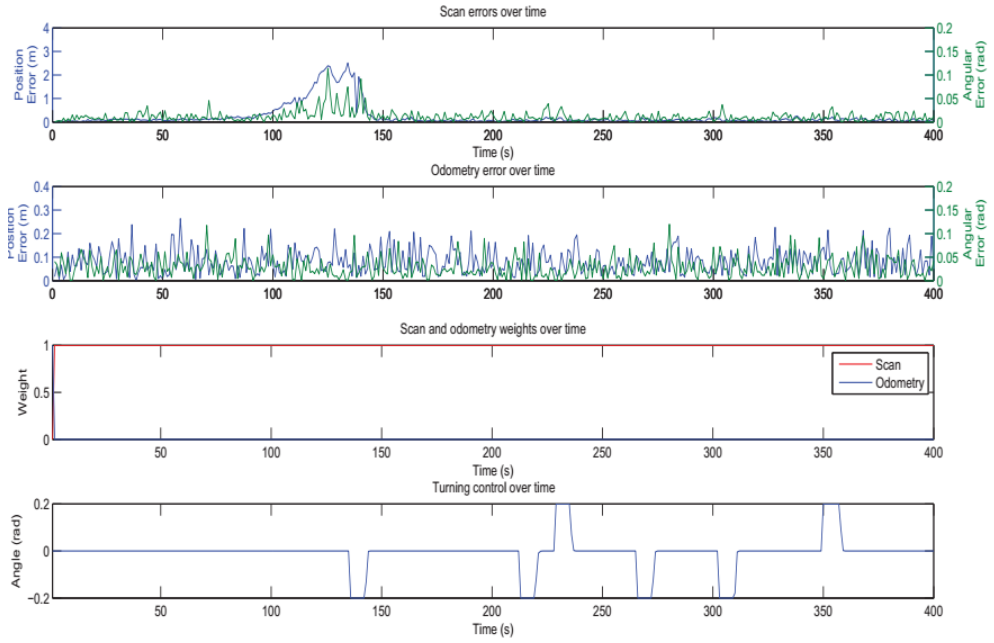


Figure 4: Visualization of the EKF landmark based SLAM

The last two Figure 4f and Figure 4e show us the results of our simulation. Due to the probabilistic framework, the results every time are slightly different, which is one of interest characteristic. However, after everything, these results prove us one of very promising SLAM approaches.
The next two pages are statistic graphical representation of parameters overall the SLAM simulation. As it can be seen from the Figure 5 together with the Figure 6, there is a place where the position error becomes relatively high, which is relevant to the strange red orbit in the Figure 4e. This error in

position due to sensor scan is the reason why everything is shifted in the Figure 4c. At this point, the error in angular orientation is also showed as high peaks.

Personally, we think these error is taken place in the left-bottom corner along the settled path of the robot. Suitable explanation may be because this place has only 4 available landmarks in the 180 degrees front of the robot, and the left-bottom corner is the first corner from the start of the simulation when the robot still has not sufficient data to become enough consistent for a good turning. One of evidence supporting for this explanation is the overshoot of standard deviation in y direction in the Figure 6, in combination with the path of the robot, where the left-bottom corner is the first corner in its way, which suddenly changes the robot's y coordinate while the state vector x is still naive for this skilled estimation We can also see, after the first failure, even though everything is shifted but all the relative mapping results are good enough for preventing this kind of error from occurring at the following corners



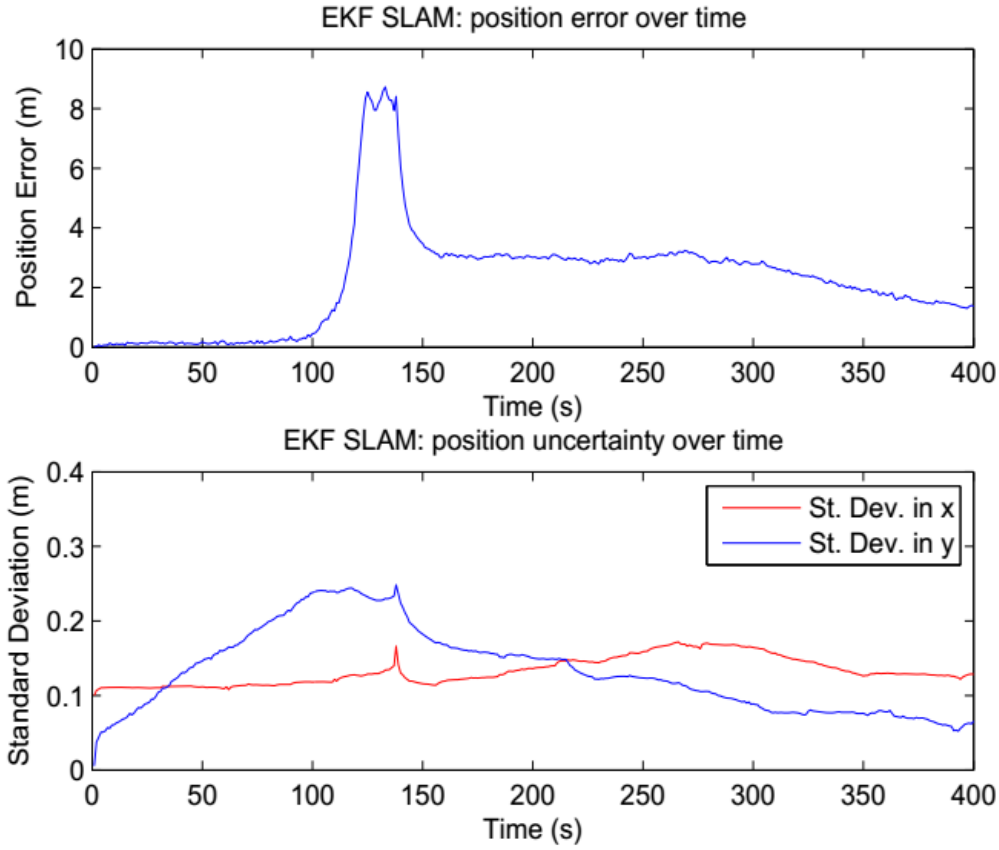


Figure 5: A statistic graph of estimation after simulation process

Modelling and Visualization

GMapping is a highly efficient Rao-Blackwellized particle fler to gradually build grid maps from laser scanner data. GMapping is probably most used SLAM algorithm, which is currently the standard algorithm on the PR2. Further information can be found at [2]. Here in ROS, we have slam_gmapping package which contains a wrapper around gmapping, providing us with SLAM capabilities

4. Reinforcement Learning concepts in Search-based Planning

In this session, we explore the reinforcement learning based on the agent can be able to maximize the total of reward. In Figure 4.1 shows the cumulative reward according to each step and can be able to formalize by equation:

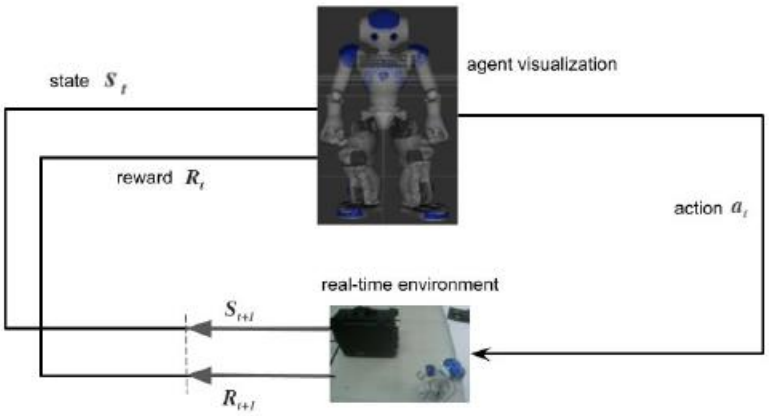


Figure 4.1: [16, 22] Reinforcement Problem: Defining the interaction between the agent and Maze Environments.

From maze environments the autonomous vehicles or robotics receive the initial state: S_0, and it can also take the action A_0. In order to transition from one state to to other states, the agent can move to other state S_1, and one reward will be received according to this state: R_1. Or it generally can be written by

$$G_t = \sum_{k=0}^{T} \quad R_{t+k+1} \qquad (4.2)$$

There is essentially to discount the rewards called gramma γ, and it belong distance: 0 and 1as the conditions, or shortly $(0 \leq \gamma < 1)$

- For short-term rewards if there is the grammar $\gamma$is small, then discount will be big
- For long-term rewards if there is the grammar $\gamma$is large, then the discount will be small.

So we can rewritten the formalization from (4.2)

$$G_t = \sum_{k=0}^{\infty} \quad \gamma^k R_{t+k+1} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \qquad (4.3)$$

where $\gamma \in$.

There are currently two kinds of tasks representing in reinforcement learning problems, involved episode which is created the lists of states S_t, actions A_t, rewards R_t, new states S_{t+1}, and usually it started initial point and ended point is a terminal state, and continuous tasks, which are no existing the terminal state.

There are currently two methods for learning, called Monte Carlo and Temporal Difference Learning accordingly by representing the two equations below:

$$V^{\pi}(s_t) \leftarrow V^{\pi}(s_t) + \alpha(G_t - V^{\pi}(s_t)) \quad (4.4)$$

$$V^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, a, s') + \gamma(V^{\pi}(s'))] \qquad (4.5)$$

Where $\alpha$is a learning rate, $V(S_t)$is the previous estimation, $R_{t+1}$is the next of reward, and

basically the Temporal Difference function is defined by the part: $R_{t+1} + \gamma V(S_{t+1})$. One of most challenges in reinforcement learning is trying to balance between the exploration and exploitation (trade-off). While exploitation is expect to maximize the totals of rewards, the exploration is to discover more information about environments. Usually, we use the full knowledge which is the dynamic programing as the methods for solving the optimal policy or optimal value functions using bellman optimality equations, it is formalized by:

$$V^{\pi}(s_t) \leftarrow \mathbb{E}_{\pi}\big(R_{t+1} + \gamma V^{\pi}(s_{t+1})\big) \qquad (4.6)$$

Accordingly, there exists two model approaches for reinforcement learning that lists in Table 1.1 below illustrated the involving models model-based and model-free learning

Model-based and Model-free
Learning action models by using the transition probabilities. For instance, it is usually to use the dynamic programming for solving approximate problems. On the other hand, model-free can learn directly action models without confident to extract model of action. By the way, Q-learning is used to solve the model-free learning [17]
For datasets, while model-based learning often require to have larger data for training, model-free learning needs less data set
For example, Greedy Adaptive Dynamic Programing is used to learn the optimal policy without evaluate the policy fixing.

Table 1.1: Model-based learning and Model-free learning

There are three approaches for solving the reinforcement learning
Value Iteration: In the reinforcement learning contexts, the value iteration is to return the maximum expecting from agent at each state.

$$v_{\pi}(s) = E_{\pi}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \ \vee S_t = s\big] \quad (4.7)$$

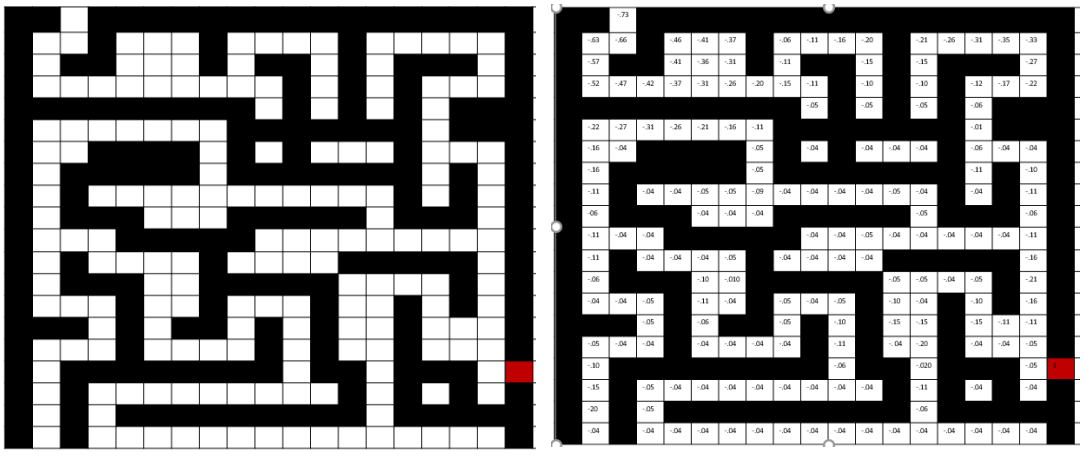


Figure 4.2: a) initial maze environment and b) generation directions for maze environments

In figure 4.2, we implemented the sampling example bug algorimth [24] by using the markov

decision processing for maze environments (20x20) extended from [24] [9]. It shows the optimized path from optimal policy.

5. Reinforcement Learning concepts in Search-based Planning

5.1. General landmark based SLAM

There are three approaches for solving the reinforcement learning

Value Iteration: In the reinforcement learning contexts, the value iteration is to return the maximum expecting from agent at each state.

There are currently studied based on research-based planning and replanning at [17] applying to autonomous system and robotics. In this experiences, we tried two experiences not only in simulation but also executed in real-time environments.

Let's consider experiments in Figure 5.1. In the Left side, it illustrated the initial point with regardless 8-nearest points as representing on [17]. Next one,

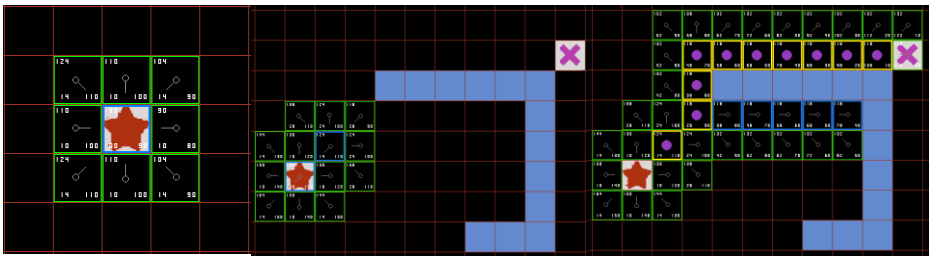


Figure 5.1: Simulation Path-Planning: [LEFT] - Start to explore the nearest all of nodes from initial point. [CENTER] - It is continue to extract the maze environments based on representing the goal which is determined at beginning and until attached the first cell of the obstacle. [RIGHT] -

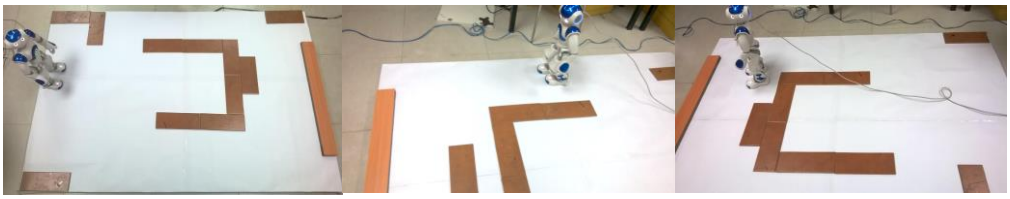


Figure 5.2: Real-Time Autonomous System and Robotics in A* Algorithm: [LEFT] - Initial Position. [CENTER] - Trajectory Generation from Structure Maze Environments. [RIGHT] - Reached the Final Position.

5.2. SLAM-based on Unstructured Environments

In this experience, we show the concepts using gmapping implemented in ROS (Figure 5.3). The system architecture can be mapped the real-time environments using reference [2]. For motion control, we refer to standard wheel encoder systems[8] [3], [12].

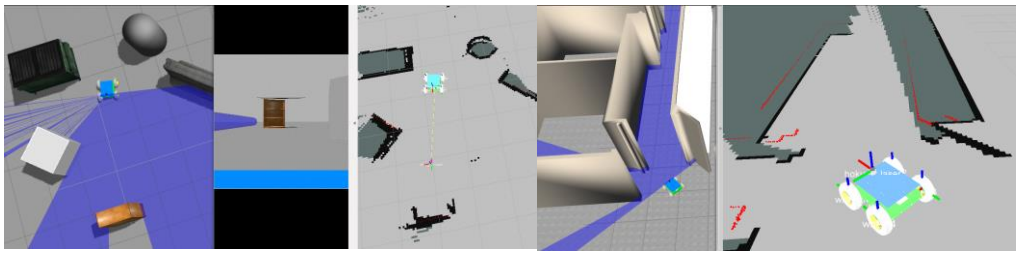


Figure 5.3: SLAM-based on uncertainty environments applied to Vehicle Mobility.

5.3. General landmark based SLAM

As previous section, we choose the Q-learning as the option of solving the reinforcement learning. Algorithm 3 is described the Q-Learning which can be able to apply autonomous system. For more detail, please refer this work [16] which was integrated the reinforcement learning.

Algorithm 3: Q-Learing()

begin
  Initialize $Q(s,a)$ arbitrarily, $\forall s \in S, a \in A(s)$;
  /* Repeat for every episode forever */
  while $True$ **do**
    Initialize S;
    while $True$ **do**
      if *Terminal-state is* $False$ **then**
        Choose A from S using a policy ($\epsilon - greedy$ policy);
        Take action $A, S \leftarrow S'$, check reward R;
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1}$ $+\gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t))$;
      else
        if *Terminal-state is* $True$ **then**
          $Q(S_t, A_t) \leftarrow Q(S_t, A_t)$ $+\alpha(R_{t+1} + 0 - Q(S_t, A_t))$;

In Figure 5.4

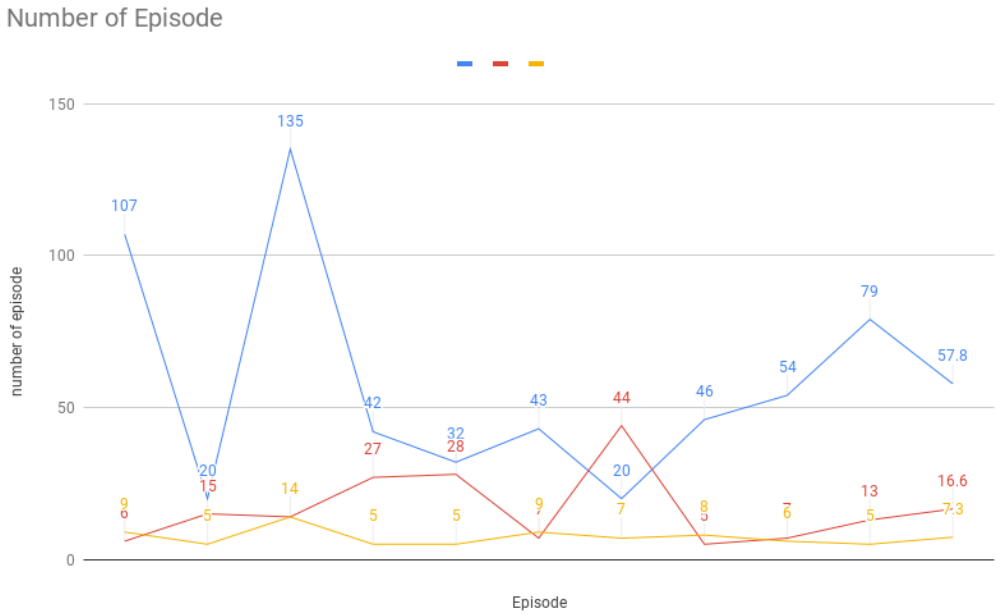
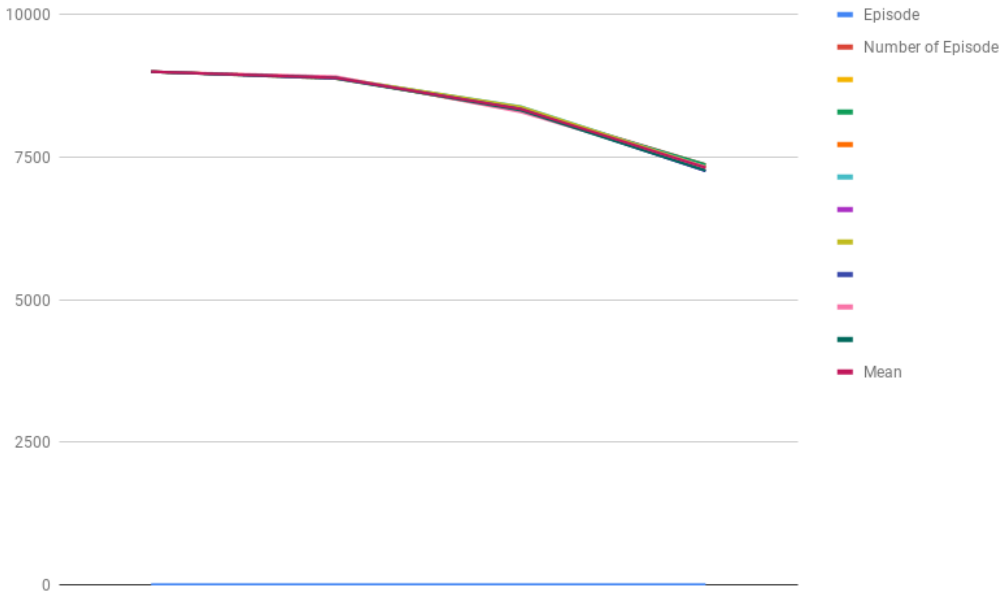


Figure 5.4:



6. Conclusions

In summary, we understand the principles of linear and nonlinear problem in probabilistic approaches based on using Kalman Filter and Extended Kalman Filter respectively as facing the uncertainty problems. We also show the modeling by representing the xml format and demo the visualization for autonomous systems. It also illustrated how to deal with uncertainty representation, which is the most important thing in challenge of self-driving vehicles in advanced technologies. At the same time we explore the search-based planning based on understanding the backtracking of trajectory generation in not only A* algorithms but also D* with Reset, and both applied to shortest

path planning. The sample implementation of reinforcement learning was selected to optimization of state-of-the-art dealing the path planning in advanced autonomous vehicles.
In the future, we integrated the deep reinforcement learning for solving the decision-making and deep learning approach to solve the localization and mapping in order to increasing the performance..

References

1. Bens, K. and von Puttkamer, VIEWEG+TEUBNER. E. (2009). Autonomous Land Vehicles.
2. Claessens, R., Mueller, Y., and Schnieders, B. (2013). Graph-based simultaneous localization and mapping on the turtlebot platform.
3. Fox, D., Thrun, S., and Burgard, W. (2005). Probabilistic Robotics. The MIT Press.
4. of Alabama in Huntsville, T. U. (2009). Probability. [online] http://www.math.uah.edu/stat/prob /Probability.html. Accessed on June 18th, 2015.
5. Chen, G., editor (2003). Approximate Kalman Filter. World Scientific.
6. K.Chui, C. and Chen, G. (2001). Kalman Filtering with Real-Time Applications. Springer.
7. SiMon, D. (June 2001). Kalman filtering. Embedded Systems Programming.
8. Khiem N. Doan, An T. Le, Than D. Le, Nauth Peter. Swarm Robots Communication and Cooperation in Motion Planning. In: Dan Zhang ,Bin Wei, editors.Mechatronics and Robotics Engineering for Advanced and Intelligent Manufacturing. Springer, Cham; 2016.p.191-205.DOI:htps:/ doi.org/10.107/978-3-319-3581-0_15
9. Søren Riisgaard , Morten Rufus Blas . SLAM for Dummies. The dummies (2011).

10. A. T .Le, M. Q. Bui,T. D. Le and N. Peter, 'D* Life with Reset:Improved Version of * Lite for Complex Environment' In:FirstIE International Conference Robotic Computing(IRC); Taichung, Taiwan .IEEE; 2017 .p.160-163 .DOI: 10.1109/IRC.2017.52

11. Souza, C. (2010). Random sample consensus (ransac) in c sharp. Blog. [online] http://crsouza.com/2010/06/random-sample-consensus-ransac-in-c/. Accessed on July 2rd, 2015.

12. Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S. (2005). Principles of Robot Motion-Theory, Algorithms, and Implementation. The MIT Press

13. Hoi V. Nguyen, Than D. Le, Dung D. Huynh, Peter Nauth, Forward kinematics of a human-arm system and inverse kinematics using vector calculus, 14th International Conference on Control, Automation, Robotics and Vision, IEEE Xplore, 2016. DOI: 10.1109 /ICARCV.2016.7838641

14. Durrant-Whyte, H. and Bailey, T. (2008). Simultaneous localisation and mapping (slam): Part ii state of the art

15. Durrant-Whyte, H., Fellow, IEEE, and Bailey, T. (2008). Simultaneous localisation and mapping (slam): Part i the essential algorithms

16. Than D. Le ; An T. Le ; Duy T. Nguyen, Model-based Q-learning for humanoid robots, 18th International Conference on Advanced Robotics (ICAR), IEEE Xplore, 2017. DOI: 10.1109/ICAR.2017.8023674

17. An T. Le and Than D. Le 'Search-Based Planning and Replanning in Robotics and Autonomous Systems', Book: Advanced Path Planning for Mobile Entities, Rastislav Róka, IntechOpen, DOI: 10.5772/intechopen.71663

18. Volodymyr Mnih and Koray Kavukcuoglu and David Silver and Andrei A. Rusu and Joel Veness and Marc G. Bellemare and Alex Graves and Martin A. Riedmiller and Andreas Fidjeland and Georg Ostrovski and Stig Petersen and Charles Beattie and Amir Sadik and Ioannis Antonoglou and Helen King and Dharshan

Kumaran and Daan Wierstra and Shane Legg and Demis Hassabis, 'Human-level control through deep reinforcement learning' Nature, pages: 529-533, 2015. DOI:10.1038/nature14236

19. Than D Le, Dang T Huynh, Huy V Pham, 'Efficient Human-Robot Interaction using Deep Learning with Mask R-CNN: Detection, Recognition, Tracking and Segmentation', ICARCV 2018: 162-167.

20. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, 'Proximal Policy Optimization Algorithms', arXiv, 2017.

21. David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, L Robert Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, Demis Hassabis, 'Mastering the game of Go without human knowledge ', Nature, 2017. DOI:10.1038/nature24270

22. Richard S. Sutton and Andrew G. Barto, 'Reinforcement Learning: An Introduction', Second Edition, MIT Press, Cambridge, MA, 2018.

23. Quan H Nguyen, Trinh NP Tran, Dung D Huynh, An T Le, Than D Le, 'Real-Time Localization and Tracking System with Multiple-Angle Views for Human Robot Interaction', The First IEEE International Conference on Robotic Computing (IRC), IEEE Xplore, 2017.

24. TD Le, DT Bui, VH Pham, Encoded Communication based on Sonar and Ultrasonic Sensor in Motion Planning, IEEE SENSORS 2018.