MTECH PROJECT PROGRESS REPORT

Mapping, Localisation and Navigation of a Delivery robot in Dynamic indoor Envirnment

by

Chanchal Rai
(2017med1005)

Under the supervision of

Dr. Ekta Singla & Dr. Shashi Shekhar Jha

DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY ROPAR

SEPTEMBER 2021

Certificate

I am submitting my MTech Project progress report. I certify it to be my original work, and referred sources(journal, books, conference proceedings, manuals, etc.) have been duly acknowledged. This work contains all the details of the work done by me this semester.

Date : September 2010

CHANCHAL RAI
(2017med1005)

Chanchal rai has worked sincerely under our supervision for aforementioned work. We have gone through report and it is up to our expectation. He has appropriately reflected his work through this report.

Date : September 2021

Dr. Ekta Singla & Dr. Shashi Shekhar Jha

Acknowledgement

I express my gratitude to my research supervisor Dr. Ekta Singla and Dr. Shashi Shekhar Jha, for allowing me to research. They provided invaluable guidance through their dynamism, motivation, vision, and sincerity. It has deeply inspired me. They have guided me through a strategy to carry out the research and present the research works as understandably as possible. It was a great pleasure and honor to work and study under their guidance. I am incredibly grateful for what they have offered me. I would thank them for their friendship, empathy, and great sense of humor. I would also like to thank Mr. Anubhav Dogra (PhD scholar), who assisted me whenever I needed help.
I also thanks my parents for their love, care and prayers. I am blessed to have such parents who always stand with me whenever i needed help.

Chanchal Rai
IIT Ropar
Date: September 2021

Contents

List of Figures

List of Algorithms

Nomenclature

1. **Symbols**
   - t $\rightarrow$ time
   - x_t $\rightarrow$ state vector at time t
   - u_t $\rightarrow$ control vector at time t
   - z_t $\rightarrow$ measurement vector at time t
   - p $\rightarrow$ probability
   - A,B,C $\rightarrow$ Constant
   - $v\bar{a}r$ $\rightarrow$ predicted variable
   - $bel(x_t) \rightarrow p(x_t|z_{1:t}, u_{1:t})$: probability of x given z and u each at time t
   - ϵ, η $\rightarrow$ Error term
   - μ $\rightarrow$ mean
   - Σ $\rightarrow$ Covariance
2. **Abbreviations**
   - LGVs $\rightarrow$ Laser Guided Vehicles
   - AGVs $\rightarrow$ Automated Guided Vehicles
   - SLAM $\rightarrow$ Simultaneous Localisation and Mapping
   - KF $\rightarrow$ Kalman Filter
   - EKF $\rightarrow$ Extended Kalman Filter
   - PF $\rightarrow$ Particle Filter
   - RRT $\rightarrow$ Rapidly exploring Random Tree
   - LIDAR $\rightarrow$ Laser Detection and Ranging
   - RGB-D $\rightarrow$ Red Green and Blue - Depth
   - ROS $\rightarrow$ Robot Operating System
   - RViz $\rightarrow$ ROS visualizer

Chapter 1

Introduction

Simultaneous Localisation and Mapping (SLAM)[4] is a basic requirement for autonomous vehicles. Mapping means to draw the map of the environment as close as possible to reality. Localisation means the robot will have knowledge about relative position with obstacles in the environment. Both tasks are done simultaneously with additional data of movement of a robot using motor to accurately draw map of environment. This whole process come under SLAM. After mapping is done and the robot is localized, the job of the robot is to travel autonomously from its position to target safely, without any collision within a time frame. SLAM along with path planner is implemented in a robot to make it sufficient to be working autonomously.

1.1 Motivation

In the current world, which is heading towards industry 4.0 but for delivery, Laser guided Vehicles (LGVs)[6] are still used in the industry. The industry is going towards automation, but LGVs are guided by laser reflector techniques. Reflectors had to be installed at the proper position for a smooth transition. Before LGVs, line follower robots were used[7]. Both these systems are not feasible for customization. It requires different setup for different functions which increases the running cost and runtime. To deal with such a problem, AGVs are necessary. AGVs need a map of the environment and they can travel without colliding. It can be given a destination to which it can safely complete within a limited time cycle. For such delivery usage, if there can be a vehicle that can work safely in an indoor environment (hospital, academic building, etc.), it will draw the map of the building without any human aid and deliver anywhere on that map. Mapping of the indoor environment is a challenging task. There are doors and windows that can be opened or closed and this will significantly affect the map creation and movement of robot. Creation of map in dynamic environment is an another challenge. The peoples walking in the building can hide some section in the map and thus will not be visible to the robot. This will affect the map creation. These are the challenges that have to be countered.

1.2 Background

There are algorithms that can solve that AGVs purpose of delivery. Independent works have been reported in the areas of mapping algorithms and path planning algorithms but development on collective work is limited and it has received huge attention in recent times.

SLAM is the way of building a map of an unknown environment while localizing itself in the map. Localization means information of relative distance with obstacles and orientation in the surrounding. To build the map of the environment, robots have to visualize every corner of the map, and this information is provided by sensors. It will be an ideal task if the true odometry of the robot is known at every point of time, i.e., ideal working of all the sensors, motor, and camera. however, the sensor data is mostly loaded with noise to some degree. So, there needs to be some estimation required for navigation instead of absolute measurement. The absolute building of a map is impossible hence slam uses a probabilistic approach[4]. The probabilistic approach is more robust in real scenario. They can provide better measurement in terms of mean and variance and can be close to real behavior. They can handle uncertainty with probability weightage and thus can neglect the effect of random data. Since an accurate system is a point data, but the probability provides a function that is close to real data. So there is a requirement of comparatively faster machines. There is also a limitation of function which have to generate data as close as to real measurement. These all can be satisfied with Recursive Bayesian Estimation[8][9] at cost of computational inefficiency which is reduced using various proposed algorithms.

Once the map is generated, there is need of path to reach at destination avoiding collision. Path connection between source and destination cannot be straight. There need to be collision avoidance maintaining safer distance with obstacles. Path generation is done as per node by node basis. Those generated nodes are connected straight or staggered depending on algorithm. No direct connection can lead to a path so node generation is performed on random basis. There is path planning algorithms generated on random basis and all nodes are connected to each other to make complete path. That path is optimized to make it shorter and varies from one algorithm to other. All algorithms are mentioned.

Chapter 2

Literature Review

We are dealing with probability estimation. The term that can be used for basic understanding of terminology are -

- **SLAM** - SLAM is way of building a map of unknown environment while localizing itself in the map.
- **Localisation** - Localisation means information of relative distance with obstacles and orientation in the surrounding.
- **State** - State is a collection of all the aspects of robot. It stores information about orientation relative to global frame. It also consists of information about its location,velocity and velocity of its joints and . It is represented as x_t i.e. x at time t.
- **Belief** - Belief is information of state of environment known by robot. In robotics, probabilistic method of belief is implemented using conditional probability. $bel(\bar{x}_{t-1})$ denotes the belief of posterior without calculating measurement at t^{th} time. $bel(x_t)$ denotes the belief of posterior after t^{th} measurement update.

$$\bar{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t})$$

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$$

2.1 Simultaneous Localisation and Mapping

Recursive Bayesian filter[9] is method to create map of the surrounding and Recursive Markov Localisation[10] (derived from Bayesian filter) is used to derive about the localisation of the robot in the environment.

2.1.1 Bayes Filter

The Bayesian filter[8][9] uses two types of equations: Gaussian systems and discrete(particle) systems. Bayes filter for Gaussian system is presented in Algorithm(1) and for discrete system in Algorithm(2). There are mainly two steps in the Bayes filter[4]: prediction and correction. Prediction step (3rd line in algorithm1) is determined by integral over all previous state x, before time t. To represent it in recursive form current prediction is integral of control u_t and $bel(x_{t-1})$ over x_{t-1} to x_t. Correction step (4th line in algorithm1) corrects the current state of the robot by probability with which measurement has been observed.

Algorithm 1 Algorithm Bayes Filter [4]

1: $AlgorithmBayesfilter(bel(x_{t-1}, u_t, z_t)$:
2: **for** all x_t **do**
3: $\quad \bar{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx$ ▷ prediction
4: $\quad bel(x_t) = \eta.p(z_t|x_t)bel(x_t)$ ▷ correction
5: **end for**
6: $returnbel(x_t)$

Algorithm 2 Algorithm Discrete Bayes Filter[4]

1: $AlgorithmDiscreteBayesfilter(p_{k,t-1}, u_t, z_t)$:
2: **for** all k **do**
3: $\quad \bar{p}_{k,t} = \sum_i p(X_t = x_k|u_t, X_{t-1} = x_i)p_{i,t-1}$ ▷ prediction
4: $\quad p_{k,t} = \eta p(z_t|C_t = x_k)\bar{p}_{k,t}$ ▷ correction
5: **end for**
6: $returnp_{k,t}$

2.1.2 Markov Localisation

Markov Localisation[10] is derived from Bayes filter thus the algorithm is similar to Bayes filter(Algorithm 1). Additionally, it requires a map(m) as input. It also has two variants like Bayes filter: one for continuous distribution and other for discrete distribution. Algorithm for continuous distribution is called Markov Localisation(Algorithm 3) and for discrete distribution is called Grid localization(Algorithm 2) and mentioned correspondingly.

Algorithm 3 Algorithm Markov Localisation[4]

$AlgorithmMarkovLocalisation(bel(x_{t-1}, u_t, z_t, m) :$
for all x_t **do**
    $\bar{bel}(x_t) = \int p(x_t|u_t, x_{t-1}, m)bel(x_{t-1})dx$
    $bel(x_t) = \eta.p(z_t|x_t, m)bel(x_t)$
end for
$returnbel(x_t)$

Algorithm 4 Algorithm Grid Localisation[4]

$AlgorithmGridLocalisation(p_{k,t-1}, u_t, z_t, m) :$
for all k **do**
    $\bar{p_{k,t}} = \sum_i p_{i,t-1}motion\_model(mean(x_k), u_t, mean(x_i))$
    $p_{k,t} = \eta.measurement\_model(z_t, mean(x_k), m)$
end for
$returnp_{k,t}$

2.1.3 Kalman Filter(KF) and Extended Kalman Filter(EKF)

Kalman filter[11] was the first filter that uses Bayes filter technique for solving SLAM problem. It uses linear Gaussian systems. The belief, $bel(x_t)$ is expressed using mean μ_t and covariance Σ_t at time t. The current state in KF is represented as linear gaussian systems:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

The current measurement in KF: $z_t = c_t x_t + \zeta_t$
KF algorithm is not attached due to its bad efficiency and lesser applications. Algorithm for its descendant i.e. EKF, is included.

EKF[4][5] can solve non linear models (Algorithm 5) which KF can not handle. EKF key idea is to linearize non-linear models. For linearization, it makes use of first order of Taylor Expansion. The current state:

$$x_t = g(u_t, x_{t-1}) + \epsilon_t$$

The current measurement: $z_t = h(x_t) + \zeta_t$ This function is replaced by first order of Taylor expansion to linearize it. Linearization of posterior state and measurement is-

$$\begin{aligned} g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1}) \\ &= g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \end{aligned}$$

$$\begin{aligned} h(x_t) &\approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)(x_t - \bar{\mu}_t) \\ &= h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) \end{aligned}$$

These non linear functions go through linearization process where first order Taylor expansion is used. After Linearization, KF is applied on resultant linear gaussian system.

Algorithm 5 Algorithm Extended Kalman Filter[5]

1: $ExtendedKalmanfilter(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t):$
2: $\bar{\mu}_t = g(\mu_t, \mu_{t-1})$
3: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4: $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$ ▷ $K_t is Kalman Gain$
5: $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$
6: $return \mu_t, \Sigma_t$

2.1.4 Extended Kalman Filter Localisation

EKF is (Algorithm 5) for drawing map of a given environment. EKF Localisation is based on Markov Localisation(Algorihtm 3). EKF localisation is of two types: with known correspondence and unknown correspondence. Localisation with known correspondence is like hypothetical situation because no measurement can be ideal. Unknown correspondence is real life situation and its implementation is very much less. Its algorithm is is very much slow, so not included. Improved algorithm is attached later.

2.1.5 Particle Filter

Particle filter[4] is non-parametric implementation of Bayes filter. It uses discrete bayes filter(Algorithm 2). A random set of samples determines bel(x). Particle filter (Algorithm 6) represent the posterior as bel(x) and all posterior distribution samples are called particles and are denoted by

$$X_t := x_t^{[1]}, x_t^{[2]}, ...x_t^{[M]}$$

M is the number of particles. Weights are updated after every likelihood estimation. Estimation is done after every control u_t, done for state of each M particles and x_{t-1} are updated and z_t is correspondingly measured. On the correction step, the weights get updated depending on the measurement for different particles. More the particles, better the state estimation and slower the calculation. Error of approximation decreases to zero as the number of particles representing posterior goes to infinity. Each particle denotes a state at time t.

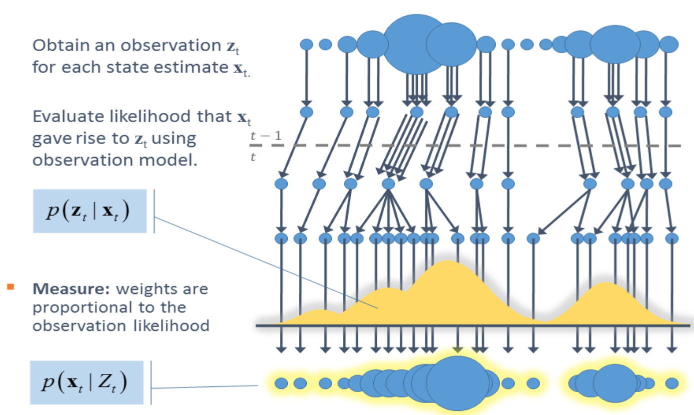

Figure 2.1: Weightage updation of particles[1]

Algorithm 6 Algorithm Particle Filter[4]

Algorithm Particle filter (X_{t-1}, u_t, z_t) :
$\bar{X}_t = X_t = \phi$ ▷ normal distribution
for $m = 1 to M$ **do**
    sample $x_t^{[m]}$ $p(x_t|u_t, x_{t-1}^{[m]}$
    $w_t^{[m]} = p(z_t|x_t^{[m]})$
    $X_t = X_t + < x_t^{[m]}, w_t^{[m]} >$
end for
for $m = 1 to M do$ **do**
    draw i with probability $\propto w_t^{[i]}$
    add $x_t^{[i]}$ to X_t
end for
return X_t

2.1.6 Adaptive Monte Carlo Localisation(AMCL)

MCL[4] is a localisation algorithm that uses Grid Localisation(Algorithm 4). It is totally based on particle filter(Algorithm 6). Algorithm for MCL is not attached but its updated version AMCL is attached. Advanced version of MCL is AMCL that uses randomization to better distribute particles (Algorithm 7).

Algorithm 7 Algorithm Adaptive Monte Carlo Localisation[4]

$AlgorithmMonteCarloLocalisation(X_{t-1}, u_t, z_t)$:
w_{slow}, w_{fast}
$\bar{X}_t = X_t = \phi$
for $m = 1toM$ **do**
    $samplex_t^{[m]} = sample\_motion\_model(u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} = measurement_{m}odel(z_t, x_t^{[m]}, m)$
    $X_t = X_t + < x_t^{[m]}, w_t^{[m]} >$
    $w_{avg} = w_{avg} + \frac{1}{M} w_t^{[m]}$
end for
$w_{slow} = w_{slow} + \alpha_{slow} w_{avg} - w_{fast}$
$w_{fast} = w_{fast} + \alpha_{fast} w_{avg} - w_{slow}$
for m=1 to M **do**
    if with probability max(0.0, 1.0 - w_{fast}/w_{slow}) **then**
        add random pose to X_t
    else
        draw i $\in$ 1,.....,N with probability $\propto w_t^{[i]}$
        add $x_t^{[i]}$ to X_t
    end if
end for
$returnX_t$

2.1.7 FastSLAM

FastSLAM[2] takes advantage of which other algorithms had not even considered (Algorithm 8). It uses Rao-Blackwellization[12] to exploit dependencies between variables. This simple conditional probability theorem states that if $p(b|a)$ can be calculated efficiently, represent p(a) only with samples and compute p(b—a) for each sample.

$$p(a, b) = p(b|a)p(a)$$

It takes the assumption that for the current pose x_t, the robot only depends on last control i.e. u_{t-1} and last pose x_{t-1}. Factorization of SLAM posterior

$$p(x_{0,t}, m_{1,N}|z_{1,t}, u_{1,t}) = p(x_{0,t}, |z_{1,t}, u_{1,t}) \prod_{n=1}^{N} p(m_i|x_{0,t}, z_{1,t})$$

It factors out robot path x^t and each of N landmarks m_n. State of each M particles are defined for $N + 1$ variables. Each landmark is represented with mean $\mu_{n,t}$ and variance $\Sigma_{n,t}$.

$$x_t^{[m]} = (x^{t[m]}, \mu_{1,t}, \Sigma_{1,t}, ..., \mu_{n,t}, \Sigma_{N,t})$$

For localisation purpose, it use AMCL (Algorithm 7).

Algorithm 8 Algorithm FastSLAM[2]

$AlgorithmFASTSLAM(c_t, u_t, z_t, X_{t-1}):$
for $k = 1toN$ **do** ▷ loop over all particles
    $Let < x_{t-1}^{[k]}, < \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]} > ... >$
    $x_t^{[k]}\ p(x_t|x_{t-1}^{[k]}, u_t)$ ▷ sample pose
    $j = c_t$
    if $featurejneverseenbefore$ **then**
        $\mu_{j,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$ ▷ intialised mean
        $H = h^{'}(\mu_{j,t}^{[k]}, x_t^{[k]})$ ▷ calculate jacobian
        $\Sigma_{j,t}^{[k]} = H^{-1}Q_t(H^{-1})^T$ ▷ initialise covariance
        $w^{(k)} = p_0$
    else
        $\mu_{j,t}^{[k]}, \Sigma_{j,t}^{[k]} = EKFUpdate()$ ▷ landmark updation
        $w^{[k]} = |2\pi Q|^{\frac{-1}{2}} exp\frac{-1}{2}(z_t - \hat{z}^{[k]})^T Q^{-1}(z_t - \hat{z}^{[k]})$
    end if
    for $allunobservedfeaturedj^{'}$ **do**
        $< \mu_{j',t}^{[k]}, \Sigma_{j',t}^{[k]} > = < \mu_{j',t-1}^{[k]}, \Sigma_{j',t-1}^{[k]} >$
    end for
end for
$X_t = resample(< x_t^{[k]}, < \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]} >, ..., w^{[k]} >_{k=1,2,...N})$
$returnX_t$

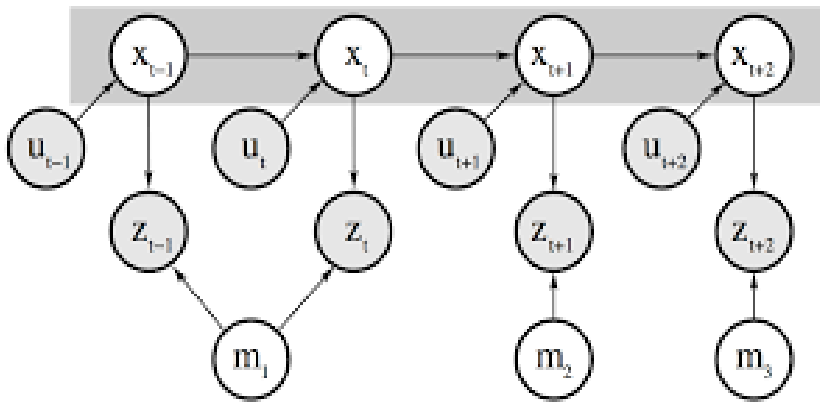

Figure 2.2: State variable at time t depends on control at t-1 and at state variable t-1 [2]

2.2 Comparison of Different SLAM Filter

To represent a comparative reference for different SLAM filters, this section presents a prominent point related to robustness, efficiency and ROS compatibility.

1. EKF
   - Robustness → Low.
   - Efficiency → Complexity is of order $O(N^{2.8}) \approx O(N^3)$. Due to its high complexity, it is only suitable for smaller maps.
   - ROS compatibility → yes, mrpt_ekf_slam_2d package use this algorithm.
2. Particle Filter
   - Robustness → Low.
   - Efficiency → same as EKF.
   - ROS compatibility → yes, mrpt_localisation package use this algorithm.
3. FastSLAM
   - Robustness → More than EKF.
   - Efficiency → Complexity is of order O(MLog(N)).

- ROS compatibility → yes, gMapping package use this algorithm.

N stands for no. of landmarks and M is no. of particles. EKF algorithm is less robust than FastSLAM algorithm. FastSLAM is a lot faster than EKF and particle filter. In terms of localisation, AMCL is gold standard. EKF localisation lacks in accuracy and efficiency, so never used.

2.3 Path Planning

After drawing of map and localization of robot, robot should know the path to reach to destination. The path is decided using random generation of nodes and nodes are connected to reach to destination. Connection of nodes depends on algorithms. Three of the most efficient algorithms are discussed below in detail.

2.3.1 Rapidly exploring Random Tree (RRT)

Path planning is the algorithm that decides what path to follow to reach a destination. It works on random node generation basis. Following three algorithms are presented with proper explanation and diagram. More latter algorithms are efficient compared to earlier ones. RRT[3][13] builds tree using random sampling in search space(Algorithm 9). Branches start from an initial node and are expanded randomly in search space to reach the predetermined target. Iteration starts with selection of a random node in search space. Feasibility of the state depends on where the random node is generated. If node is generated on obstacle, node generation is repeated otherwise most near nodes from tree are selected to establish link. There is a predetermined distance range in which the nearest node is to be selected. If the random state is in the range, then link is connected using the boolean collision check. It grows pixel by pixel and follows unoccupied pixels. This check ensures obstacle-free feasible connectivity. If the random state lies outside of that range, then that random state is discarded, and a new random state is generated again. This process is continued until the predefined time interval or the predefined number of iterations.

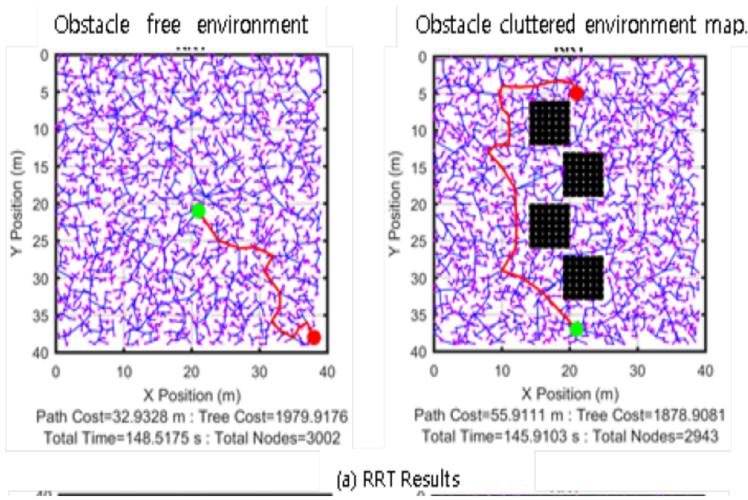

Figure 2.3: Path prediction using RRT [3]

Algorithm 9 Algorithm RRT[3]

$T = (V, E) \leftarrow RRT(z_{(init)}$
$T = InitializeTree();$
$T = add\_Node(\phi, z_{init}, T);$
for $i = 0toi = Ndo$ **do**
    $z_{rand} \leftarrow generate(i);$
    $z_{nearest} \leftarrow most\_near(T, z_{rand});$
    $(z_{new}, U_{new}) \leftarrow Steer(z_{nearest}, z_{rand});$ ▷ provide control to reach new node
    if $Obstacle(z_{new})$ **then**
        $T = add\_Node(z_{min}, z_{new}, T);$
    end if
end for
$returnT$

2.3.2 RRT* (RRT updated)

RRT*[3][13] is an descendant of the RRT algorithm. It follows all properties of RRT and also introduces new features (Algorithm 10). It introduces near neighbor search operation. This search operation find parents for the newly generated random node. Among the possible parents, available within the a circle of radius

$$k = \gamma \frac{log(n)}{n}^{\frac{1}{d}}, where$$

d $\rightarrow$ dimension of search space
$\gamma \rightarrow$ planning constant, depends on arrangement in environment.

It can also be said that new node is built within a radius of k. New random node is then connected to best fit parent to minimize connection cost and to maintain connectivity. RRT* adds one more feature on RRT that is path optimization. As the number of nodes increases, the path to target is optimized. This makes the path from source to destination less jaggy and shorter.

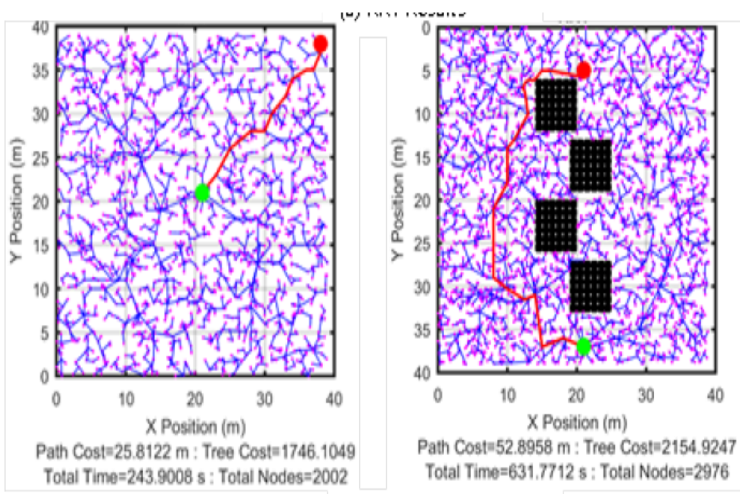

Figure 2.4: Path prediction using RRT* [3]

Algorithm 10 Algorithm RRT*[3]

$T = (V, E) \leftarrow RRT * (z_{(ini)}$
$T = InitializeTree();$
$T = add\_Node(\phi, z_{init}, T);$
for i=0 to N **do**
  $z_{rand} \leftarrow genrate(i);$
  $z_{nearest} \leftarrow most\_near(T, z_{rand});$
  $(z_{new}, U_{new}) \leftarrow Steer(z_{nearest}, z_{rand});$
  if $Obstacle(z_{new})$ **then**
    $z_{near} \leftarrow find\_Near(T, z_{new}, |V|)$
    $z_{min} \leftarrow find\_Parent(z_{near}, z_{nearest}, z_{new});$
    $T = add\_Node(z_{min}, z_{new}, T);$
    $T \leftarrow Rewire(T, z_{near}, z_{min}, z_{new});$
  end if
end for
$returnT$

2.3.3 RRT* Smart (RRT* updated)

RRT*-Smart[3] Algortihm (11) inherits properties from RRT*. The algorithm is provided for same. It is an extension to the existing RRT* (Algorithm 10) by providing a path optimization approach after a path is found. It removes redundant nodes from the found path and also makes path improvement by identifying beacon nodes (Landmark or anchor nodes). It also has a different sampling technique rather than random sampling. It uses a biasing radius like RRT*, but this radius is set around selected beacons. Once it reaches the destination node, it performs path optimization process and, in the process, generates new beacon nodes. So, overall this algorithm accelerates path convergence and lowers path cost.

Algorithm 11 Algorithm RRT*-Smart[3]

$T = (V, E) \leftarrow RRT * {-Smart}(z_{(ini)}$
$T = add\_Tree();$
$T = add\_Node(\phi, z_{init}, T);$
for $i = 0toN$ **do**
    if $i = n + b, n + 2b, ...$ **then**
        $z_{rand} \leftarrow generate(i, z_{beacons};$
    else
        $z_{rand} \leftarrow genrate(i);$
    end if
    $z_{nearest} \leftarrow most_near(T, z_{rand});$
    $(z_{new}, U_{new}, T_{new}) \leftarrow Steer(z_{nearest}, z_{rand});$
    if $ObstacleFree(z_{new})$ **then**
        $z_{near} \leftarrow find\_Near(T, z_{new}, \|V|)$
        $z_{min} \leftarrow find\_Parent(z_{near}, z_{nearest}, z_{new});$
        $T = add\_Node(z_{min}, z_{new}, T);$
        $T \leftarrow Rewire(T, z_{near}, z_{min}, z_{new});$ ▷ node connection
    end if
    if $initialPathFound$ **then**
        $n \leftarrow i$
        $(T, directCose) \leftarrow PathOptimisation(T, z_{init}, z_{goal});$
    end if
    if $(directCostNew < directCostOld)$ **then**
        $z_{beacons} \leftarrow PathOptimization(T, Z_{init}, Z_{goal});$
    end if
end for
$returnT$

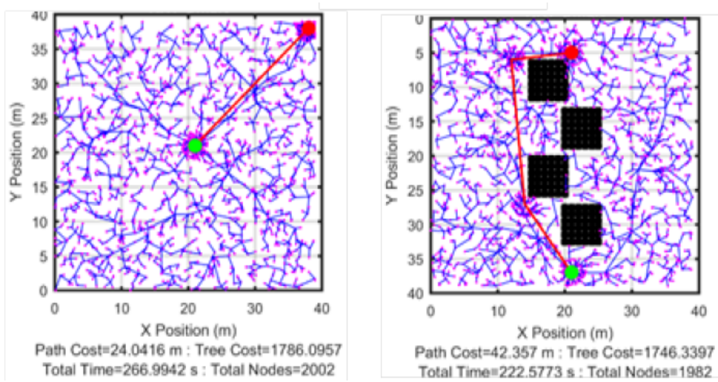

Figure 2.5: Path prediction using RRT*-Smart[3]

2.4 Comparison of Different Path Algorithms

- RRT is based on a random generation of nodes that are connected through a straight line.
- RRT* is an updated version of RRT that uses the near neighbor feature. This avoids generation impractical paths. It generates random nodes in the radius-R in vicinity of pre-generated nodes.
- RRT* Smart is an updated version of RRT*. This generates nodes similar to RRT*, but it does so only on selected nodes and has the smart feature of sampling. RRT* is the fastest and provides the most optimized path.

References

[1] particle filter image. https://towardsdatascience.com/particle-filter-a-hero-in-the-world-of-non-linearity-and-non-gaussian.

[2] Johan Alexandersson and Olle Nordin. Implementation of slam algorithms in a small-scale vehicle using model-based development, 2017.

[3] Iram Noreen, Amna Khan, and Zulfiqar Habib. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. *International Journal of Computer Science and Network Security (IJCSNS)*, 16(10):20, 2016.

[4] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[5] Gabriel A Terejanu et al. Extended kalman filter tutorial. *University at Buffalo*, 2008.

[6] Sushant S Patil. Automated guided vehicle system (agvs). 2020.

[7] Abhijit Pathak, Refat Khan Pathan, Amaz Uddin Tutul, Nishat Tahsin Tousi, Afsari Sultana Rubaba, and Nahida Yeasmin Bithi. Line follower robot for industrial manufacturing process. *International Journal of Engineering Inventions*, 6(10):10–17, 2017.

[8] Bayes Theoram investopedia. https://www.investopedia.com/terms/b/bayes-theorem.asp.

[9] Recursive Bayes Theoram wikipedia. https://en.wikipedia.org/wiki/Recursive_Bayesian_estimation.

[10] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of artificial intelligence research*, 11:391–427, 1999.

[11] Abu Bakar Sayuti H M Saman and Ahmed Hesham Lotfy. An implementation of slam with extended kalman filter. In *2016 6th International Conference on Intelligent and Advanced Systems (ICIAS)*, pages 1–4, 2016.

[12] Keith Y. K. Leung, Felipe Inostroza, and Martin Adams. An improved weighting strategy for rao-blackwellized probability hypothesis density simultaneous localization and mapping. In *2013 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pages 103–110, 2013.

[13] Fahad Islam, Jauwairia Nasir, Usman Malik, Yasar Ayaz, and Osman Hasan. Rrt*-smart: Rapid convergence implementation of rrt* towards optimal solution. In *2012 IEEE international conference on mechatronics and automation*, pages 1651–1656. IEEE, 2012.