UNIVERSITY OF CALIFORNIA
Los Angeles

This project is under review

Autonomous navigation: SLAM, Motion Planning, and Control

A project submitted in partial satisfaction
of the requirements for the degree
Master of Science in Mechanical and Aerospace Engineering

by

Aidan Kennedy

2021

ABSTRACT OF THE PROJECT

Autonomous navigation: SLAM, Motion Planning, and Control

by

Aidan Kennedy
Master of Science in Mechanical and Aerospace Engineering
University of California, Los Angeles, 2021

This project presents an overview of autonomous navigation, divided into three sections: Simultaneous Localization and Mapping (SLAM), Motion Planning and Control, and demonstration of successful autonomous navigation on a real world mobile robot using ROS. The three paradigms of SLAM (Kalman Filters, Particle Filters, and Graph-based SLAM) will be discussed. Recent work pushing SLAM during general spatial AI will briefly covered. Motion planning algorithms of A^* and RRT^* will be discussed, as well as trajectory planning algorithms such as DWA and TEB. Finally, ROS and Gazebo are used to implement SLAM with the packages of GMapping and Cartographer, in conjunction with TEB to demonstrate real world autonomous navigation.

The project of Aidan Kennedy is approved.

TBD

University of California, Los Angeles

2021

TABLE OF CONTENTS

1 Introduction **1**

2 Overview of SLAM **3**

2.1 SLAM problem statement 4

2.1.1 Probability Density Functions 6

2.2 SLAM Anatomy 8

2.3 Hardware 9

2.4 Online Filters and Offline Smoothing 10

2.5 Map representations 12

2.5.1 Occupancy Grid Maps 13

2.6 Bayesian Filters 13

2.6.1 Kalman Filters 14

2.6.2 Particle Filters 16

2.7 Graph-based SLAM 18

2.7.1 Creating the graph 18

2.8 Conclusion 21

3 Overview of Motion Planning and Control **22**

3.1 Markov Decision Processes 23

3.2 Global Path Planner 23

3.2.1 Shortest Path Algorithms 24

3.3 Local Path Planner 26

3.3.1 Motion Models . . . 27
3.3.2 DWA . . . 31
3.3.3 Timed Elastic Band . . . 31
3.4 Conclusion . . . 31

4 ROS and Gazebo . . . 33
4.1 ROS structure . . . 33
4.2 SLAM packages . . . 34
4.3 ROS and Gazebo . . . 34
4.3.1 Overview of Gazebo . . . 35
4.4 ROS packages . . . 35
4.4.1 Navigation Stack package . . . 36
4.4.2 AMCL package . . . 37
4.4.3 tf package . . . 37
4.5 Robot Description . . . 38
4.6 Tuning the navigation stack . . . 38
4.6.1 Cost Maps . . . 39

5 Data Collection and Analysis . . . 41
5.1 Hardware . . . 41
5.2 SLAM . . . 41
5.2.1 Gmapping . . . 42
5.2.2 Cartographer . . . 42
5.3 Navigation . . . 43

6 Conclusion of work **48**

6.1 Future Steps 48

References **50**

LIST OF FIGURES

2.1 Depiction of the SLAM Problem [7] 5
2.2 Front and Back End SLAM [5] 8
2.3 Depiction of Online SLAM [35] 11
2.4 Depiction of Offline SLAM [35] 12
2.5 Example depiction of Graph based SLAM [18] 20

3.1 High level depiction of trajectory rollout [1] 27

4.1 Depiction of navigation stack [2] 37

5.1 Map produced from GMapping 43
5.2 AMCL initial particle distribution 44
5.3 AMCL updated particle distribution after driving 45
5.4 Navigation in rviz 46
5.5 Navigation through doorway 47

LIST OF TABLES

CHAPTER 1

Introduction

Computation is pervasive. From the first programmable computer of the ENIAC [15] to modern supercomputers, computer performance has increased over one hundred quadrillion times. This incomprehensible, rapid explosion in computational capability has allowed the extensive and pervasive automation of everyday tasks we see today. Most of the computers we interact with are confined to cyberspace, distinct from the real world we occupy on a day to day basis. The field of robotics, which aims to have systems perceive, think, and act in the real world is rapidly blurring this distinction.

At the current moment, there are very few real world robotic systems that interact with the world in a natural, useful way. They are confined to narrow tasks, with specific built hardware and software that struggle to adapt to new challenges outside the scope of design. Most currently deployed robotic systems, such as robotic arms in manufacturing, are restricted to stationary bases. Although efficient at the specific task, stationary bases do not generalize to more complex automation tasks, such as moving items across large distances, or exploring unknown terrain. Robust and efficient mobile bases that can autonomously navigate the environment are required to realize the full potential of real world robotic systems in everyday life and beyond. The successful deployment of robotic systems in manufacturing, agriculture, health care, deep-sea expedition, and space exploration must be able to navigate the environment autonomously for extended periods of time.

In this project, the problem of autonomous navigation will be divided into two sec-

tions: Robot Perception, and Robot Motion. Robot Perception can be pragmatically solved through a wide range of techniques within the field of Simultaneous Localization and Mapping (SLAM). Robot Motion can be solved by applying search algorithms and control policies to the mapped space. Together, these two areas of engineering provide a pathway toward fully autonomous mobile robots. The final section will demonstrate real world deployment of an autonomous robot.

This project is aimed at first year graduate or senior level undergraduate students entering the world of autonomous navigation for the first time. The field is complex, highly interdisciplinary, and rapidly evolving. It is impossible to cover all areas of the field, but it is hoped that the reader gathers a firm understanding on the theoretical foundations of SLAM, a working knowledge of how to implement these algorithms in ROS, and a direction of where to continue learning.

CHAPTER 2

Overview of SLAM

The primary objective of SLAM is to build a map of the environment, and estimate it's pose within that environment. As an active area of research, there is no single solution for any given mobile robotic system and environment. The earliest attempts to map an environment utilized Kalman Filters [10] to estimate the posterior pose in an online algorithm. By the 1990's, Extended Kalman Filters and Particle Filters dominated the field of SLAM [5], demonstrating the first successful SLAM algorithms. Although pragmatic, these online filtering techniques are generally too inefficient to map large spaces. By the early 2000's, the field shifted from advancing online state estimation algorithms to establishing offline graph optimization techniques, known as Graph-based SLAM. In contrast to the filtering techniques that solve the online SLAM problem, Graph-based SLAM exploits the entirety of information collected by constructing a graph, and then employing optimization algorithms to produce a maximum likelihood estimate of the SLAM state.

SLAM is a challenging problem for a multitude of reasons. The easier problem of pure localization within a provided map is demanding, but can be still be solved quite easily. The other problem of pure mapping given perfect localization is not a realistic option. In such a world where perfect odometry existed, it would be trivial to establish a map. Thus, one of the primary difficulties of the SLAM problem is the fact that the robot most localize itself within the environment, and also generate a map of the environment without knowing where it is. The sheer size of the hypothesis space, noise from sensors

and actuators, uncertainty in motion models, and computational constraints make SLAM extraordinarily challenging.

In the larger context of autonomous navigation, there is debate over the necessity of SLAM. Depending on the goal of the engineer, constructing a large, ever increasing, high fidelity map of the environment may be critical, or a resource burden for the embedded system that must compute and store the map. Many practitioners of the field only aim to establish high fidelity localization within the immediate environment to allow precise trajectory planning and control, and do not aim to store the entire map in memory. In particular, the field of computer vision is leading the way towards more natural image processing, allowing robots to reason about 3-dimensional spatial constraints at a high level of semantic abstraction similar to how humans do. Nonetheless, it is critical to understand the basic foundations of SLAM that employ traditional model based approaches to solving the state estimation problem. Doing so will provide clarity for where recent breakthroughs in computer vision transform the field.

2.1 SLAM problem statement

The objective of SLAM is to estimate both the robot's pose, $\mathbf{x_t}$, and the map of the environment, $\mathbf{m}$. In the case of 2-dimensional navigation, $\mathbf{x_t}$ can be fully described by a three dimensional vector consisting of two variables in the x, y plane, and an additional variable of rotation: $\mathbf{x_t} = (x_t, y_t, \theta_t)$. The map is assumed to be static, and is not a function of time. Recent advancements in SLAM that handle dynamic environments will not be considered here. The map can be represented as either a sparse matrix of landmarks, where landmarks are denoted as $\mathbf{l_i}$, or as a dense representation in a discretized grid. The former, referred to as feature-based mapping is typically more efficient. The latter, referred to as volumetric mapping is more widely employed due it's natural overlap to path planning problems. The robot has an incoming stream of

odometry, $\mathbf{u_t}$, as well as an incoming stream of observation data, $\mathbf{z_t}$. The odometry information directly estimates $\mathbf{x_t}$, and the observation data provides spatial information of the environment. For convenience, $\mathbf{X_t}$, represents the full SLAM state, defined as $\mathbf{X_t} = (\mathbf{x_t}, \mathbf{m})$.

Shown below is a depiction of the SLAM statement in the form of a factor graph. Here the robot is traversing along some environment left to right. The cyan circle nodes represent the pose of the robot $\mathbf{x_t}$, the blue square nodes represent the environment landmarks $\mathbf{l_i}$, and the black dots connecting each pose and landmark encode the likelihood factor between the two nodes. The exact details of the factor graph representation will not be covered in this project, but it is useful to note the relationships between pose and landmark nodes. As the robot drives through the environment, it gathers an increasing number of measurements to a particular landmark, and can thus reduce the uncertainty of landmark locations. This in turn reduces the uncertainty of pose estimation over time.

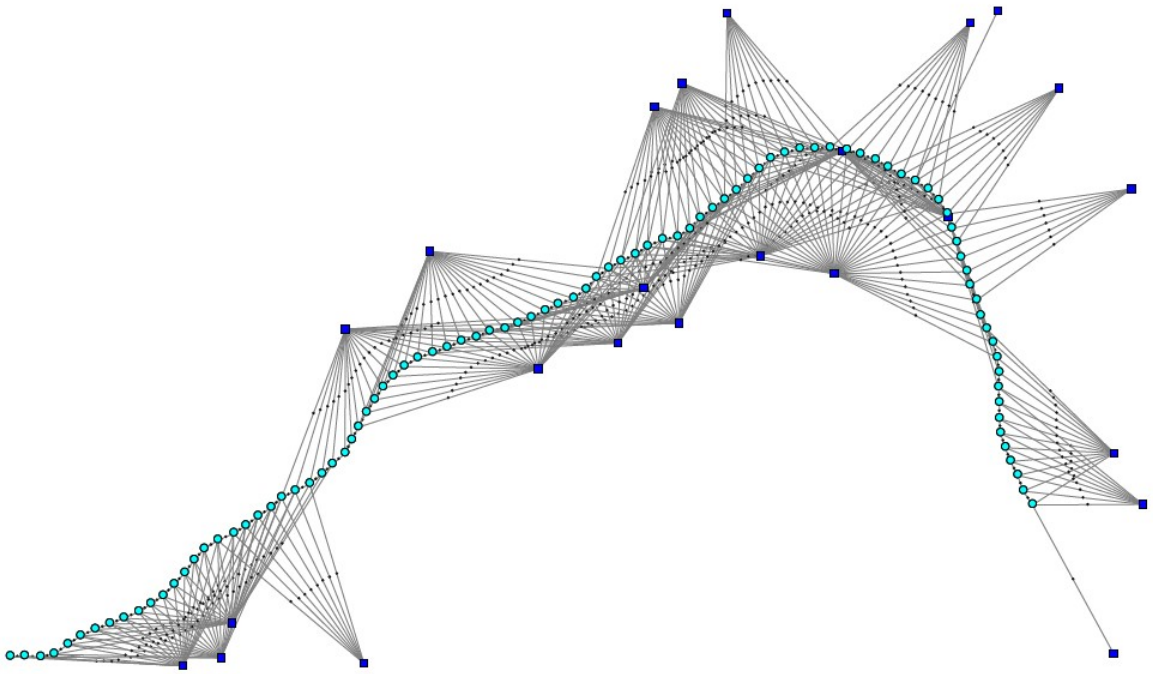

Figure 2.1: Depiction of the SLAM Problem [7]

2.1.1 Probability Density Functions

Deterministic equations are not suitable for real world robotics problems that must take into account noise and uncertainty in the system model and measurements. As such, probability distributions are computed and sampled from to capture the uncertainty in perception and motion. Bayesian inference is a common framework employed to reason about state estimation problems. In the most general sense, Bayesian probabilities take into account the prior probability of a distribution, and then update the probability distribution as new updates arrive. All SLAM algorithms aim to produce either a maximum a posterior estimation (MAP), or maximum likelihood estimation (MLE) of the full SLAM state $\mathbf{X_t}$. It is useful to note that MLE is special subset of MAP where the prior is considered to be uniform. The full SLAM problem statement can now be most generally defined as

$$\mathbf{X^{MAP}} = \underset{\mathbf{X}}{\operatorname{argmax}} P(\mathbf{X}|\mathbf{z}, \mathbf{u}) \tag{2.1}$$

Put simply, this equation states to find the maximum posterior estimation of the SLAM state $\mathbf{X_t}$ given an observed data stream $\mathbf{z_t}$ and $\mathbf{u_t}$. Maximizing this probability density can be complex and requires numerous assumptions to allow a tractable solution, but the basic idea is quite intuitive.

As a state estimation problem, it is necessary to define both a motion model to predict the evolution of the SLAM state $P(\mathbf{X_t}|\mathbf{X_{t-1}}, \mathbf{u_t})$, and a measurement model to predict the incoming data streams $P(\mathbf{z_t}|\mathbf{X_t})$. And to make the problem tractable, it is necessary to specify the structure of the probability density functions. Many PDFs can be applied, but most SLAM techniques specify a multi-variate Gaussian PDF. Although restrictive and idealistic, this assumption allows tractable solutions. A general multi-variate Gaussian PDF can be represented as

$$\mathcal{N}(\Theta; \mu, \Sigma) = \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\Big(-\frac{1}{2}(\Theta - \mu)^T \Sigma^{-1} (\Theta - \mu) \Big) \tag{2.2}$$

where Θ represents the variables of interest, μ represents the mean of the distribution, and Σ represents the covariance matrix.

2.1.1.1 Evolution of the SLAM State

It is now useful to define exactly how the SLAM state $\mathbf{X_t}$ will evolve over time. Again, it is necessary to define state estimation models that predict the evolution of the state in order to update the Bayesian probability.

The evolution of the SLAM state can be defined as

$$\mathbf{X_t} = g(\mathbf{X_{t-1}}, \mathbf{u_t}) + \epsilon_t \tag{2.3}$$

where $g(\cdot)$ is the nonlinear model of the process, ϵ_t is a zero-mean Gaussian PDF specifying the process uncertainty. This can be interpreted as simply computing the next state X_t according the function $g(\cdot)$, and then adding some uncertainty to account for imprecise model dynamics and previous state estimation error.

In a similar fashion, the incoming data stream of measurements can be defined as

$$\mathbf{z_t} = h(\mathbf{X_t}) + \delta_t \tag{2.4}$$

where δ_t is a zero-mean Gaussian PDF specifying the measurement noise, and $h(\cdot)$ is the nonlinear measurement function. This can be interpreted as computing the next measurement state $\mathbf{z_t}$ according to the function $h(\cdot)$, and then adding some uncertainty to account for imprecise model dynamics and previous state estimation error. It should be noted for clarity that the $\mathbf{z_t}$ is a vector of individual measurements $\mathbf{z_t} = \{\mathbf{z_t^1}...\mathbf{z_t^k}\}$.

The functions g and h still need to be defined, and will be discussed in later sections.

2.2 SLAM Anatomy

The SLAM algorithm is typically divided into two entangled parts know as the front-end and back-end. The front-end of SLAM is responsible for extracting meaningful features from the incoming data stream $\mathbf{z_t}$. The primary difficulty lies in discriminating between data points when performing data association. Iterative Closest Point (ICP) is a popular algorithm to perform data association on 3-dimensional point-clouds produced from LIDAR data. The back-end of SLAM is responsible for constructing a map from the features collected. The back-end does most of the heavy lifting of the algorithm, and is the focus of most modern research in SLAM. Together the two ends work in a feedback loop, specifically Loop Closure, to construct a globally consistent map of the environment. Loop closure works by identifying when a previously seen SLAM state $\mathbf{X_t}$ is experienced again, and then updating the entire SLAM state $\mathbf{X_{1:t}}$ to produce a more likely probability distribution. It is a critical feature of modern SLAM techniques, and allows the creation of globally consistent maps.

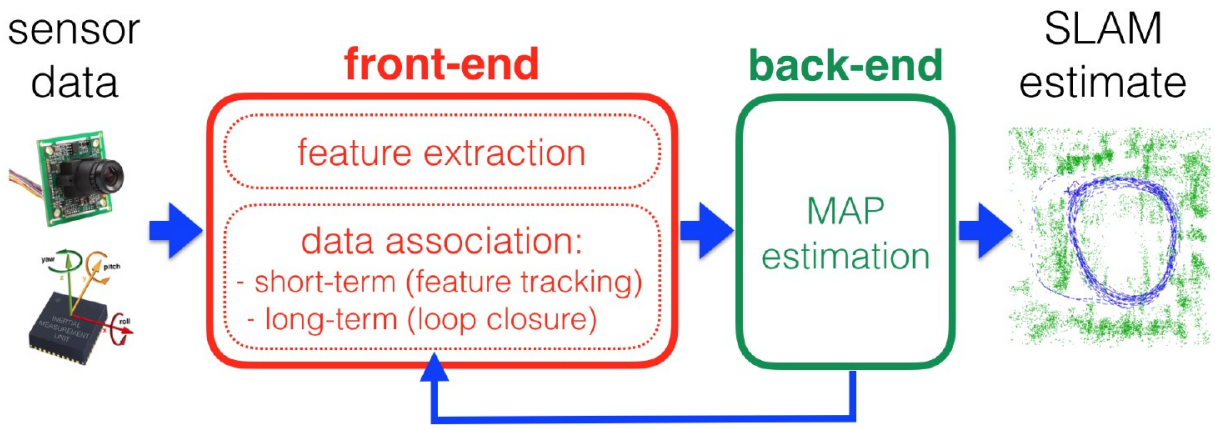

Figure 2.2: Front and Back End SLAM [5]

2.3 Hardware

SLAM can be performed with many different sensor configurations. It is hardware agnostic, and only requires some form of vision, and some form of odometry. Vision is usually achieved through cameras, ultrasonic sensors, or laser range finders (LIDAR). LIDAR, which operates by pulsating electromagnetic radiation and measuring the time of flight between the object and base station, is a popular choice for many systems. In the case of wheeled robots, odometry can be achieved through simple wheel encoders. More precise odometry estimates can be produced by integrating Inertial Measurement Unit (IMU) data.

Within the context of SLAM, there are two primary categories of hardware. (1) LIDAR based SLAM: Due to the rapid, precise, and robust measurement capabilities of lasers, it is currently the most common form of sight used within SLAM [37]. The drawbacks of Lidar include it's high cost, large sensor size, and moving parts. (2) Vision based SLAM: This is often referred to as Visual SLAM, or vSLAM. This method employs computer vision to interpret raw pixel data from ordinary cameras. 2-D and 3-D measurements can be collected through a variety of different camera configurations (such as stereo or monocular cameras). Visual SLAM is attractive due to it's low cost, but more computationally intensive than ordinary SLAM, and currently less robust to environmental complexities. It is an active area of research, and relies on recent advancements in machine learning and computer vision for meaningful feature extraction. In terms of pure information density, the data from cameras in vSLAM is unmatched. As such, vSLAM will likely emerge as the dominant form of SLAM in the coming years.

2.4 Online Filters and Offline Smoothing

Kalman Filters and Particle Filters are specific implementations of online filtering techniques. They are generally easy to implement, and attractive for mobile robots mapping small areas. Graph-based SLAM is an implementation of offline sparse graph optimization. The model of the environment is not constructed until after the robot has completed data collection, or when the user requests the map to be constructed. The full posterior model from the online SLAM problem statement is well suited for sparse graph optimization techniques [36], suggesting that online filtering techniques are a subset of the more general offline Graph-based SLAM. As opposed to filtering techniques that immediately use and then discard data, Graph based SLAM stores the entirety of data collected, allowing for much more efficient use of data. Both online and offline techniques allow for Loop closure, which is an algorithm that identifies a previously seen SLAM state $\mathbf{X_t}$, and then optimizes the entire SLAM state with the information to produce a better estimate. In the case of online filtering, this will lead to reduced uncertainty in landmarks. In the case of offline smoothing, this will lead to reduced uncertainty in the pose and landmarks. It is a critical feature of modern SLAM systems.

Shown below is a graphical depiction of online SLAM. It only estimates the current SLAM state, represented by the grey box around $\mathbf{x_{t+1}}$ and $\mathbf{m}$, which is a function of only $\mathbf{x_t}$, $\mathbf{u_{t+1}}$ and $\mathbf{z_{t+1}}$. All previous information has been integrated out and discarded.

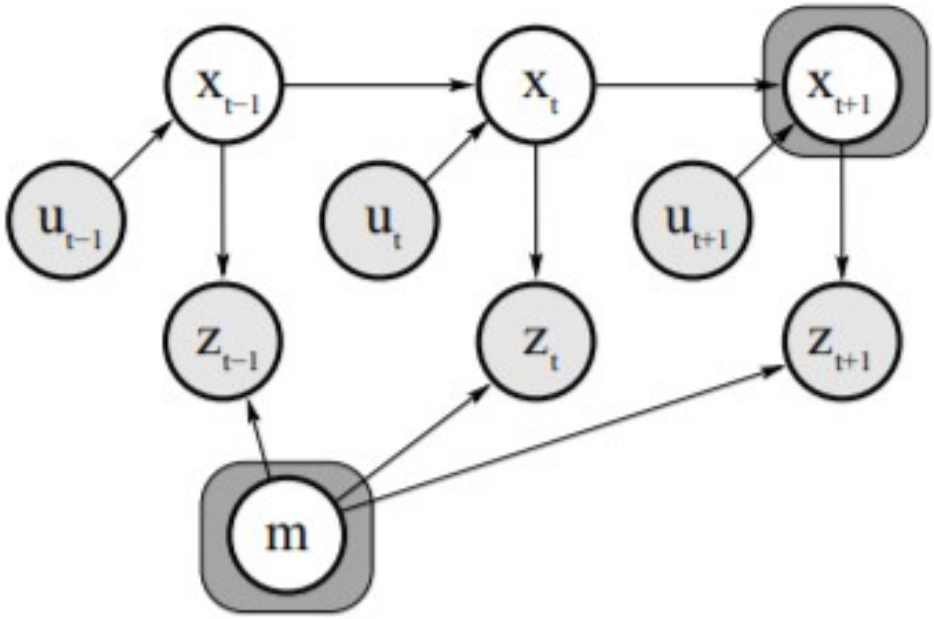

Figure 2.3: Depiction of Online SLAM [35]

Shown below is a graphical depiction of offline SLAM. It estimates the entire SLAM state, represented by the grey box around $\mathbf{x_{t-1}}$, $\mathbf{x_t}$, $\mathbf{x_{t+1}}$, and $\mathbf{m}$. It is a function of all measurements. The joint posterior of the full SLAM state is calculated, allowing new information to influence not just the current state, but entirety of past states as well. This is a powerful feature which is heavily exploited in Loop Closure to create globally consistent maps.

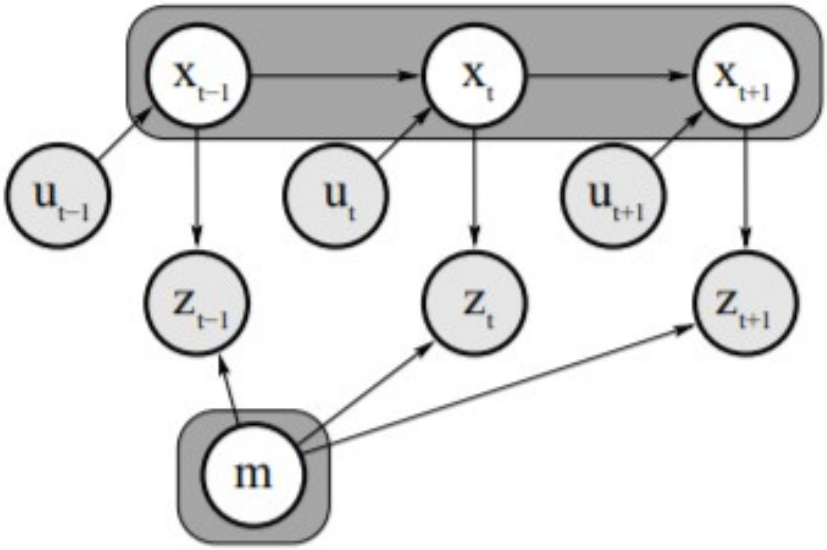

Figure 2.4: Depiction of Offline SLAM [35]

2.5 Map representations

The choice of map representation is critical to the successful autonomous mobile robot. Map representations can either be continuous or discrete approximations of the environment. Continuous representations build a topological map of the environment, and are closer to what humans utilize to reason about an environment. Although inherently difficult to apply to traditional path planning algorithms that rely on grid representations of space, they are an active area of research in Semantic SLAM. This additional layer of abstraction promises to provide efficient and robust data structures for mapping and path planning. Discrete representations discretize a space into a grid that can be easily represented as a graph. This graph is typically stored as an adjacency list or matrix, and requires large amounts of memory. Discrete representations currently dominate the field of robotics due to their ready applicability to established path planning algorithms. It has been suggested by [40] that both feature and grid based representations of maps may be required for truly robust autonomy.

2.5.1 Occupancy Grid Maps

First proposed in 1985, occupancy grid maps [29] provide a simple way to parameterize the model of the environment. They integrate naturally with grid based SLAM algorithms that seek to define probability of occupancy. A continuous space is discretized into a grid of specified resolution, where each cell of the grid represents the probability of occupancy. A value of 1 represents occupancy and is usually depicted as black, a value of 0 represents free space and is usually depicted as white. A value anywhere between 0 and 1 represents the probability of occupancy and is depicted as grey. The SLAM Algorithm is responsible for specifying the occupancy of each cell. The motion planning algorithm then uses the occupancy grid map to plan a path through the free space. Grid maps are the standard method of representing the environment due to their convenience.

2.6 Bayesian Filters

Recursive Bayesian estimation is the foundation of modern filtering techniques, and heavily used within robotics, computer science, and general state estimation problems. The Bayesian Filtering algorithm is broken up into a set of *prediction* and update equations. The *prediction* step utilizes a model of the system dynamics to predict the next measurement. In this way, it is incorporating prior information of the system. The *update* step takes a measurement of the system, compares the measurement to the prediction, and then updates the prediction to maximize the posterior estimate. In the most general sense, the filter exploits the fact that a state estimate computed from two independent sources will have a smaller variance that either of those sources alone.

Bayesian Filtering frameworks nearly always assume a Markov system to provide tractable solutions. The Markov system assumes that the future state is conditioned on only the present state, and no further back. Although not entirely faithful, the assump-

tion is both necessary to avoid needlessly complex computation and suitable for most real world systems. Real world deviations from this assumption are easily accounted for through additional error terms in the probabilistic framework. Kalman Filters and Particle Filters are specific implementations of Bayesian Filters that are used widely throughout robotics problems.

2.6.1 Kalman Filters

Dating back to the 1960's, Rudolf E. Kalman proposed a linear quadratic estimator [22] which produces the best possible state estimate for a linear system. It was developed alongside Richard S. Bucy, and first utilized in the Apollo program under the direction of Stanley F. Schmidt to solve the trajectory estimation and control problem when going to the moon [17].

Now commonly referred to as simply the Kalman Filter, it is one the most popular methods for state estimation [17], and heavily used within the realm of mobile robotics and autonomous navigation. As a linear state estimator, it must be linearized around the mean and covariance to produce reasonable results in real world nonlinear systems. When linearized, Kalman Filters are no longer an optimal linear estimator. They can suffer from poor initialization, inaccurate model dynamics, and sub-optimal linearization points. Even with the potential pitfalls, the computational efficiency of the filter, with the update equation dictated by $\mathcal{O}(n^2)$ [35] has led to nonlinear extensions such as the Extended Kalman Filter (EKF) to become the industry standard in state estimation problems [38]. It should be noted that the filter does suffer in large state estimation problems where the number of landmarks grows large.

2.6.1.1 Extended Kalman Filter Algorithm for SLAM

The Extended Kalman Filter algorithm for the the full SLAM state $\mathbf{X_t}$ is shown below. Here, the state and measurement functions are defined as follows

$$\mathbf{x_t} = g(\mu_t, \mathbf{x_{t-1}}) + \epsilon_t \tag{2.5}$$

$$\mathbf{z_t} = h(\mathbf{x_t}) + \delta_t \tag{2.6}$$

where $g(\cdot)$ is the nonlinear state transition function that takes the mean of the state μ_t and previous state $\mathbf{x_{t-1}}$, and ϵ is process noise. The function $h(\cdot)$ is the nonlinear measurement function, and δ_t is the measurement noise. Both noises are assumed to be zero-mean Gaussian processes with covariance Q_t and R_t respectively.

Algorithm 1 Extended Kalman Filter with inputs $(\mu_{t-1}, \Sigma_{t-1}, \mathbf{u_t}, \mathbf{z_t})$ [35]

1: $\overline{\mathbf{X}}_\mathbf{t} = g(\mathbf{u_t}, \mu_{t-1})$
2: $\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3: $K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$
4: $\mathbf{X_t} = \overline{X}_t + K_t(\mathbf{z_t} - h(\overline{X}_t))$
5: $\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$
6: return $\mathbf{X_t}$

The apriori state is denoted as $\overline{\mathbf{X}}_\mathbf{t}$, the posterior state of interest is denoted as $\mathbf{X_t}$, the control input is denoted as $\mathbf{u_t}$, and the measurement is denoted as $\mathbf{z_t}$ The linearization function $g(\cdot)$ is computed using a first order Taylor expansion around the mean of the posterior. It is this linearization function that differentiates the nominal Kalman Filter from the Extended Kalman Filter. The apriori covariance matrix is denoted as $\overline{\Sigma}_t$ and the posteriori covariance matrix is denoted as Σ_t. The matrix G_t represents the Jacobian of the usual state transition matrix A in classic linear system theory. The matrix R_t represents the covariance of the measurement noise. The matrix K_t is known as the Kalman Gain, and defines the weight to place on the measurement compared to the

model. The matrix H_t represents the Jacobian of the usual measurement matrix C. The matrix Q_t represents the covariance of the system process noise.

A more rigorous walk through can be found in Chapter 10 of Probabilistic Robotics [35].

2.6.2 Particle Filters

Originally called a Bootstrap Filter, the Particle Filter was first realized in 1993 [16] as a direct response to the restrictive nature of the Kalman Filter. The basis of the modern day particle filter extends back to the early 1930s [14], but it was not until the early 1990s that filter was fully implemented. By representing the probability density function with a set of sampled particles, rather than a function of the state space as in the Kalman Filter, the Particle Filter is not restricted to linear dynamics or Gaussian estimates. At the expense of computational complexity, it is a robust filter, theoretically capable of handling a state estimation problem of any underlying probability distribution.

2.6.2.1 Particle Filter Algorithm

The Naive Particle Filtering algorithm for the full SLAM state $\mathbf{X_t}$ is shown below. The particles are represented as ζ_t, where $\zeta_t = \{\mathbf{X_t^{[1]}}, \mathbf{x_t^{[2]}}, ..., \mathbf{x_t^{[J]}}\}$. There are a total of J particles, where each particle is a hypothesis for the pose and map. The variable $w_t^{[j]}$ is called the importance factor, which represents the weight assigned to each particle. Particles that are more likely to be seen from the measurement distribution will have a higher associated weight.

It should be noted that the Particle Filter algorithm as stated is woefully unwieldy due to the sheer size of the hypothesis space. Naive particle filters are generally too inefficient to solve real world state estimation problems. Robust state estimation requires a sufficiently large number of particles that quickly becomes intractable in high

Algorithm 2 Naive Particle Filter [35]

1: $\overline{\zeta}_t = \emptyset$
2: **for** $j = 1$ to J do **do**
3: sample $\mathbf{X_t}$ proportional to $p(\mathbf{X_t}|\mathbf{u_t}, \mathbf{X_{t-1}^{[j]}})$
4: $w_t^{[j]} = p(\mathbf{z_t}|\mathbf{X_t^{[j]}})$
5: $\overline{\zeta}_t = \overline{\zeta}_t + \left\langle \mathbf{X_t^{[j]}}, w_t^{[j]} \right\rangle$
6: **end for**
7: **for** $j = 1$ to J do **do**
8: draw i with probability $\propto w_t^{[i]}$
9: add $\mathbf{x_t^{[i]}}$ to ζ_t
10: **end for**
11: return ζ_t

dimensional state space. The rectify this problem, the Rao-Blackwellised Particle Filter [9] was introduced in 2000. It marginalizes the state space to a tractable size that can be successfully sampled.

In order to alleviate this problem, FastSLAM was introduced in 2002 [28]. As briefly discusses above, this variant of the Particle Filter employs a Rao-Blackwellized Particle Filter that uses particles to represent the sampled posterior over the robot poses, and Gaussian PDFs over the potential maps to minimize computational demand.

One of the critical reasons for why this approach is possible is due to the natural construction of the SLAM problem. In general, landmarks in the map are only dependent on each other as a result of the uncertainty in the pose of the robot. If the $_t$ was perfectly known, the landmarks would not be dependent on each other. This means that the errors of landmark locations for each of the particles representing the full SLAM state $_t$ are conditionally independent from each other. As such, the SLAM problem can be factored into a single estimation problem for each feature. This avoids the massively

high dimensional filter over the pose and map, and instead uses an EKF to estimate feature locations, and a separate low dimensional EKF for each individual feature.

A more rigorous walk through can be found in Chapter 13 of Probabilistic Robotics [35].

2.7 Graph-based SLAM

In contrast to the Bayesian filtering algorithms of the Kalman Filter and Particle Filter, Graph-based SLAM is an offline smoothing algorithm that utilizes the entirety of information collected to solve the full SLAM problem. One might suspect that storing and manipulating such a large amount of data might seem unwieldy or even impossible given the difficulty of online filtering techniques. However, the full SLAM problem forms a naturally sparse matrix, allowing efficient storage and manipulation that significantly outperforms an online filtering technique. First realized in its current form in 2006 [36], continued advancements in sparse matrix factorization have propelled Graph-based SLAM to become the state of the art solution method.

2.7.1 Creating the graph

As discussed previously, the full SLAM state forms a naturally sparse matrix. The online SLAM problem is really a subset of this much larger sparse matrix representation. This key insight allows the problem to be solved in an efficient, tractable manner.

As the robot moves along a path, the algorithm constructs a graph of nodes and edges. Nodes represent either the pose of the robot or landmark locations, and edges represent the spatial constraints between those nodes. These spatial constraints encode how certain the robot is about a particular measurement. As the robot is constructing the graph, it will record significantly more spatial constraints than there are nodes, leading

to an over-determined system where only approximate solutions can be obtained. As an additional layer of complexity, these constraints between nodes are highly nonlinear. The algorithm must now find the best possible approximation to this system on nonlinear constraints. As such, Graph-based SLAM can be thought of as a nonlinear least squares optimization to find the most likely configuration of nodes.

Shown below is an example of the construction of the spatial constraint between two nodes. The two nodes x_i and x_j represent either the robot's pose or a landmark location. The measurement z_{ij} represents the measurement reading of node x_j from the perspective of x_i. The information matrix Ω_{ij} represents the uncertainty of the measurement z_{ij}, and is simply the inverse of the covariance matrix. The predicted measurement $\hat{z}_{ij}$ represents the prediction of where node x_j will be from the perspective of x_i. Finally the error function $e_{ij}(x_i, x_j)$ represents the the error between the expected and actual measurement of node x_j from x_i. Thus, the spatial constraint between these two nodes are entirely characterized by the error and information matrix. This construction of edges repeats for each node along the path of the robot. It is most intuitive to reason about Graph-base SLAM as a spring-mass system. Each node can be represented a mass, and each constraint between nodes can be represented as a spring. Connections of higher uncertainty will have a soft spring, and connection of lower uncertainty will have a rigid spring. At each optimization step, the masses will be adjusted according the neighboring spring stiffness.

To utilize this information, it is necessary to calculate the probability of observing node x_j from x_j. As is standard practice it is more useful to calculate the log likelihood of the configuration so as to easily compute the negative log likelihood in later steps. It is useful to note that minimizing the negative log like-hood is identical to maximizing a function, but is much easier to compute. Combining all of the nodes together, this large optimization problem can be concisely stated as

$$\mathbf{X}^{\mathbf{MLE}} = \underset{\mathbf{X}}{\operatorname{argmin}} \sum_{i,j \in C} e_{ij}^T \Omega_{ij} e_{ij} \tag{2.7}$$

where $\mathbf{X}$ is shorthand for all the the nodes in the graph, with the explicit assumption that the map $\mathbf{m}$ is landmark based. Thus, we see that this is the same formation as the original SLAM problem statement, except now taking the prior probability to be uniform and arriving at the Maximum Likelihood Estimate.

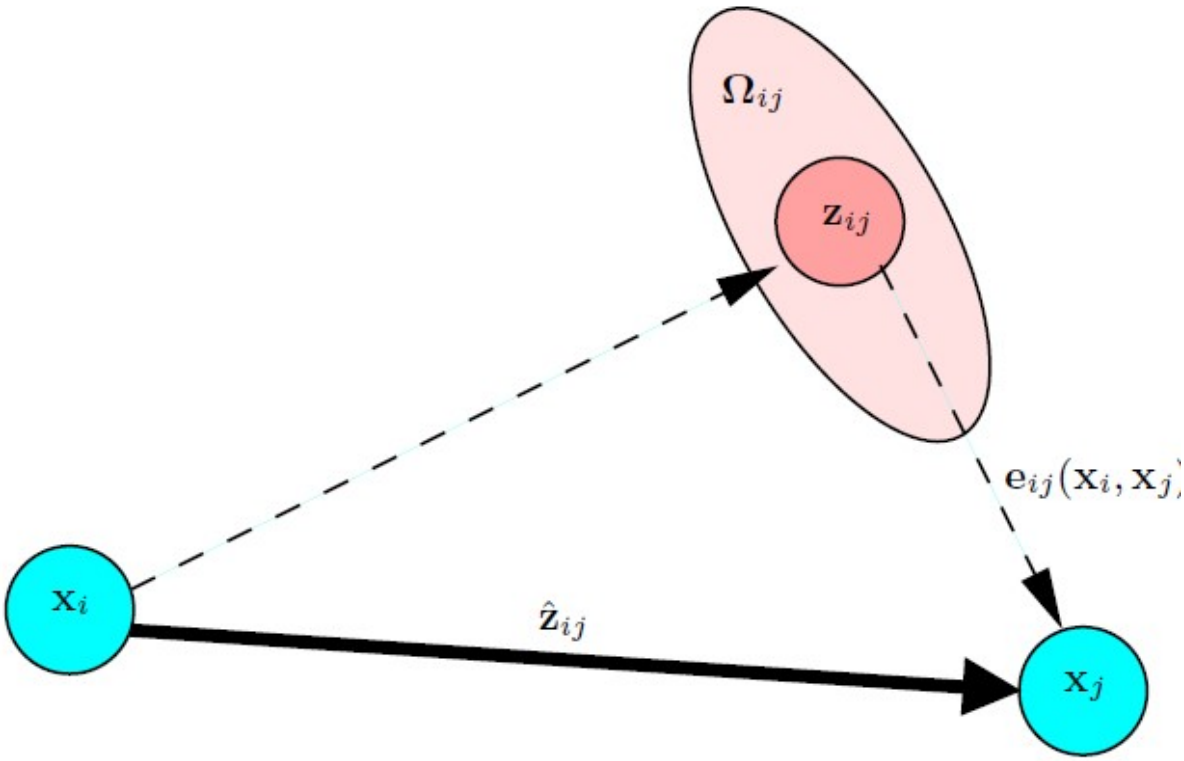

Figure 2.5: Example depiction of Graph based SLAM [18]

Although the basic idea of nonlinear least squares optimization is trivial, the sheer size of the SLAM state $\mathbf{X_t}$ makes the optimization procedure challenging. Traditional nonlinear least squares algorithms such as Gauss-Newton, Levenberg-Marquardt (LM),

QR decomposition, and more modern approaches that perform gradient descent can be applied. For a more rigorous walk-through, [18] is an excellent tutorial on how to implement the algorithm

2.8 Conclusion

The three main areas of SLAM (Kalman Filters, Particle Filters, Graph-based SLAM) have been presented. Although Kalman Filters and Particle Filter approaches may be suited for some robotics problems that require only online estimation in relatively small maps, these approaches have been eclipsed by Graph-based methods. Graph-based SLAM utilizes a sparse matrix representation of constraints, allowing the formation of extremely large maps, that can be continually refined and updated as new measurements arrive. It relies on efficient nonlinear least squares approximations. Visual SLAM, and recent breakthroughs in computer vision, are driving the field toward more robust data descriptions, promising higher levels of abstraction such as semantic reasoning. Although LiDAR based methods still have a strong role for certain situations, such as dusty or extremely high contrast environments, the reign of LiDAR based SLAM is quickly coming to an end. Visual based SLAM is dominating the field, and will only continue to as the field evolves towards general Spatial AI.

CHAPTER 3

Overview of Motion Planning and Control

The primary objective for any mobile robot is to optimally navigate to a desired final pose. Given a starting pose, a model of the environment, a model of the robot within that environment, and the final pose, the robot must be able to execute a set of trajectories to reach the final goal. Optimality is subjective and can imply metrics such a speed, efficiency, or safety depending on the application. Within this section, the terms *optimal path* and *shortest path* will be used interchangeably. As discussed in the previous section, SLAM provides practical solutions to both gathering a model of the environment, and estimating the pose of the robot within that environment. If the map is already provided, a localization technique such as Adaptive Monte Carlo Localization (AMCL) can be applied. The motion planning algorithm is then responsible to generate a set of trajectories to reach the final pose.

In the broadest sense, the motion planning algorithm must find a continuous path in the topological manifold of its configuration space (C-space). An adequate C-space for motion planning in 2-dimensions typically consists of $C = \{x, y, \theta, v, \omega\}$, where x and y are the robot's position, θ is the robots orientation, v is the translational velocity, and ω is the rotational velocity. The configuration space is broken up into two parts, C_{free} which denotes valid a valid configuration where the path must exist, and $C_{occupied}$ which denotes an invalid configuration. A valid path must traverse through C_{free}. If not already discretized, it is necessary to formulate the search space into discrete sections prior to applying search algorithms. Canonical map representations such as occupancy

grid maps are inherently discretized and require little or no preprocessing.

In general, this problem statement is too complex to be solved in real time. The configuration space defined above is far too large to be searched. More robust search spaces that include acceleration are even harder to search. It is nearly guaranteed that the search space for a given motion planning problem is intractable in its' naive form. To rectify this, the path planning problem is typically solved in two parts: A global planner and a local planner. The global planner searches a reduced C-space, and the local planner searches the full C-space over a local region. 38

3.1 Markov Decision Processes

A Markov Decision Process (MDP) is a useful framework to reason about path planning and control. First proposed in 1957 [4], they offer a framework to model uncertain, sequential decision making that can be solved via dynamic programming. In the context of robotics, they are an ideal framework to specify control policies as a function of the state space. As will be discussed, global path planners search along nodes to find the optimal pathway, typically taking the best pathway from node to node. This can be easily broken down into sequential decision making process solved through dynamic programming.

3.2 Global Path Planner

The global path planner applies a shortest path planning algorithm such as A^* or RRT^* on a lower the dimension of the search space. It does this by reducing the robot to a single point in the x, y plane, thus ignoring the spatial and kinematic constraints of the actual robot. As such, it produces a quasi-optimal pathway to the final pose. It relies on the local planner to incorporate the full state to produce a viable pathway in real time.

3.2.1 Shortest Path Algorithms

Most path planning problems in robotics can be formulated as a shortest path problem. That is given a graph of nodes and vertices, a search algorithm can be applied to find the shortest path.

In general, search algorithms are divided into two categories: uniformed search algorithms that blindly attempt to reach the goal through trial and error, and informed search algorithms that incorporate prior knowledge of the environment through the use of heuristics to reach the goal faster. Due to their efficiency, informed greedy-search algorithms such as A^*, and D^* dominate the field of robotics. Monte-Carlo based search algorithms, such as rapidly exploring tree (RRT) algorithms are a recent area of research that offer computational efficiency over optimality. Probabilistic space filling can theoretically find suitable paths faster than traditional greedy-search algorithms.

The heuristic function is a method of incorporating prior information of the environment into the search algorithm. An acceptable heuristic must always overestimate the performance of the path so as to guarantee that the cheapest pathway is found. The heuristic chosen depends on the path planning problem. Typical heuristics for a 2-D occupancy grid representation of the environment are Euclidean distance or Manhattan distance estimations.

3.2.1.1 Dijkstra's algorithm

Published in 1959 [8], Dijkstra's algorithm provides a way to calculate the shortest trajectory between two points. It works by calculating the cost to each connected node, and then updating the table of stored costs if a cheaper cost is found along a new path. This cycle repeats until a pathway to the final node is found. As an uninformed search algorithm, it forms the basis of modern informed search algorithms. Both local and global path planners are rooted in Dijkstra's algorithm.

3.2.1.2 A* and D* algorithm

First introduced in 1968 [20], the $A*$ search algorithm is a direct extension of Dijkstra's algorithm. It presents a mathematical way to incorporate prior information of the environment through heuristics, allowing an approximate solution to the shortest path problem. As long as the heuristic chosen costs less than the actual cost to the path between two nodes, $A*$ is guaranteed to find the shortest path with less computation than Dijkstra's algorithm. Global optimality is not guaranteed, but the use of heuristics allows otherwise intractable search spaces to be evaluated.

The A* algorithm samples trajectories from the search space, and executes the one that minimizes the cost function as defined by $J(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the beginning node to node n, and $h(n)$ is the heuristic function that estimates the cost of the cheapest path from node n to the goal.

The D* algorithm, standing for dynamic A*, is an extension of the A* algorithm. First proposed in 1994, it improves the performance by exploiting the fact that the planner only has to plan from the point of failure to the goal, as opposed to the entirety of the path as in A* [34]. There are many variations of the algorithm, such as Focused D*, and D* Lite.

A* and D* algorithms dominate the field of path planning in robotics as they easily integrate with discretized map representation such as an occupancy grid map, are computationally efficient, and easy to implement and tune.

3.2.1.3 RRT and RRT* Algorithms

First proposed in 1998, Rapidly Exploring Random Tree (RRT) algorithms address the computational constraints of A^* and D^* through probabilistic space-filling trees [26]. They work by randomly generating points in the search space, and connecting the closest nodes until a viable pathway is found. As a fundamentally random approach to path

planning, they are ideal for high-dimensional search spaces that cannot be analyzed rigorously through greedy-search algorithms.

RRT^* is an extension of RRT that is asymptotically guaranteed to find optimal motion plan [23]. Further extensions such as informed RRT^* introduce heuristics to find an optimal pathway faster [13]. It does this by reducing the search space to a relevant subset through heuristics. It should be noted that subject to the specific constraints of the robot and environment, A^* generally producing a shorter more optimal path that RRT^* [41].

3.3 Local Path Planner

The local path planner applies a trajectory planning algorithm such as DWA or TEB to produce a viable pathway in real-time. At its core, it is an obstacle avoidance algorithm that works in conjunction with the global planner. It is responsible for navigating around static objects that were not available at the time of computing the global plan, and dynamic objects that enter and exit the pathway. It utilizes a velocity model of the robot to compute a local plan by sampling from the space of trajectories confined to a localized region. In contrast to the global planner, the local planner takes the the entire configuration space of the robot into account.

As suggested by [30], it is useful to frame local path planning algorithms as a specific implementation of Model Predictive Control (MPC). MPC is a subset of state feedback control theory that uses a model of complex dynamic systems to produce an optimized control input at each timestep. As opposed to classical control techniques such as Linear Quadratic Regulators (LQR) or Robust Control that optimize the control input across the entire design window, MPC iteratively updates the control input. As such, MPC is able to produce a local optimal control input even when the system deviates from the prediction of the system model. The optimization algorithm at each time step makes it

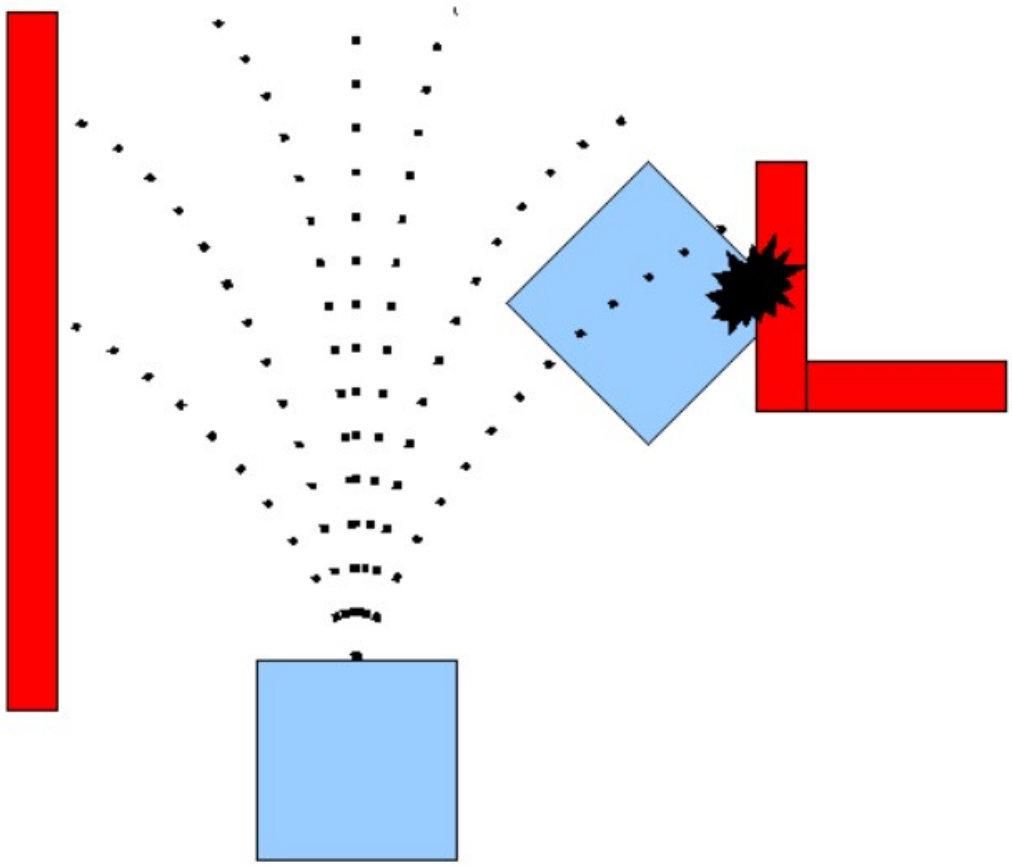

Figure 3.1: High level depiction of trajectory rollout [1]

easy to account for nonlinear dynamics. It is a very flexible control strategy that handles the uncertain nature of robotic systems. Most local path planning algorithms can be thought of as a form of MPC.

3.3.1 Motion Models

Motion models that define the state transition probability $p(\mathbf{x_t}|\mathbf{u_t}, \mathbf{x_{t-1}})$ are required for motion planning. Odometry models are typically used for estimation, and velocity models are typically used for probabilistic motion planning. In general, it is more useful to sample from the motion model as opposed to simply calculating the posterior probability. This plays nicely with Particle Filters, which are the primary algorithms used in

localization. As such, this section will show algorithms that sample from the posterior distribution.

As discussed in previous sections, the pose of the robot is denoted as $\mathbf{x_t} = (x_t, y_t, \theta_t)$.

3.3.1.1 Odometry Motion Model

The odometry motion model is used to estimate the position of the robot from internal odometry measurements, typically provided by wheel encoders. When discussing the odometry motion the control input is denoted as $\mathbf{u_t} = (\mathbf{x_{t-1}}, \mathbf{x_t})$.

The odometry motion is divided into three distinct calculations for each time step. First, the initial rotation of the robot, denoted as δ_{rot1}, is computed. Second, the translation motion, denoted as δ_{trans}, is computed. Third, the final rotation, denoted as $\delta rot2$, is computed. This sequence of rotation, translation, and final rotation is sufficient and necessary to fully describe the motion of a 2-dimensional odometry model. Using only an initial rotation angle, which may seem sufficient at first, will not be able to represent the final pose of the robot.

Algorithm 3 Sampling from Odometry Motion Model [35]

Sampling from odometry model given $(\mathbf{ut}, \mathbf{x_{t-1}})$

1: $\delta_{rot1} =$atan2$(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$

2: $\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$

3: $\bar{\theta}' = \bar{\theta} - \delta_{rot1}$

4: $\hat{\delta}_{rot1} = \delta_{rot1} - \mathbf{sample}(\alpha_1 \delta^2_{rot1} + \alpha_2 \delta^2_{trans})$

5: $\hat{\delta}_{trans} = \delta_{trans} - \mathbf{sample}(\alpha_3 \delta^2_{trans} + \alpha_4 \delta^2_{rot1} + \alpha_4 \delta^2_{rot2})$

6: $\hat{\delta}_{rot2} = \delta_{rot2} - \mathbf{sample}(\alpha_1 \delta^2_{rot2} + \alpha_2 \delta^2_{trans})$

7: $x' = x + \hat{\delta}_{trans} cos(\theta + \hat{\delta}_{rot1}$

8: $y' = y + \hat{\delta}_{trans} sin(\theta + \hat{\delta}_{rot1}$

9: $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

10: return $x_t = (x', y', \theta')^T$

The function $\mathbf{sample}(\cdot)$ returns a value from a normal distribution of zero mean and specified variance. Other probability distributions can applied if desired, although it is most common to specify a normal distribution. The parameters α_1 through α_4 capture the motion noise of the system. These parameters typically need to be determined empirically for each robot through trial and error.

3.3.1.2 Velocity Motion Model

The velocity motion model is utilized in the trajectory planner to estimate possible trajectory doing the motion planning phase. The local path planners of *DWA* and *TEB* call this velocity model for each point sampled in the search space. When discussing the velocity motion the control input is denoted as $\mathbf{u_t} = (v, \omega)$.

The velocity motion model is separated into two distinct calculated for each timestep. First, the posterior translational velocity $\hat{v}$ and rotational velocity $\hat{w}$, as well as an additional random term for robustness $\hat{\gamma}$ are computed. Second, the actual pose of the robot,

denoted as $\mathbf{x_t} = (x', y', \theta')$

Algorithm 4 Sampling from Velocity Motion Model [35]

Sampling from velocity model given $(\mathbf{ut}, \mathbf{x_{t-1}})$

1: $\hat{v} = v + \mathbf{sample}(\alpha_1 v^2 + \alpha_2 \omega^2)$

2: $\hat{\omega} = \omega + \mathbf{sample}(\alpha_3 v^2 + \alpha_4 \omega^2)$

3: $\hat{\gamma} = \mathbf{sample}(\alpha_5 v^2 + \alpha_6 \omega^2)$

4: $x' = x - \frac{\hat{v}}{\hat{\omega}} sin(\theta) + \frac{\hat{v}}{\hat{\omega}} sin(\theta + \hat{\omega} \triangle t))$

5: $y' = y + \frac{\hat{v}}{\hat{\omega}} sin(\theta) - \frac{\hat{v}}{\hat{\omega}} sin(\theta + \hat{\omega} \triangle t))$

6: $\theta' = \theta + \omega \triangle t + \hat{\gamma} \triangle t$

7: return $x_t = (x', y', \theta')^T$

Again, the function $\mathbf{sample}(\cdot)$ returns a value from a normal distribution of zero mean and specified variance. Other probability distributions can applied if desired, although it is most common to specify a normal distribution. The parameters α_1 through α_6 capture the motion noise of the system. These parameters typically need to be determined empirically for each robot through trial and error.

3.3.1.3 Measurement model

A measurement model, a model of the Laser Beam model in the case of LiDAR, is required for localization. This model computes the probability density $p(\mathbf{z_t}|\mathbf{X_t})$. Due to the immense computational demands, there are two main paradigms for calculating this probability distribution. The first method uses a physics based approach to crudely capture critical properties of the laser beam, and account for noise through a serious of error parameters. This is computationally challenging, and requires an accurate model with well tuned error parameters. The second method does away with a physics based model, and employs a simple likelihood field to estimate the likelihood of detecting objects at specified x, y coordinates. Although adhoc, it typically performs better than

the physics based model, and is easier to compute.

3.3.2 DWA

The dynamic window approach (DWA) is a common implementation of the local path planner. First proposed in 1997, [12] it incorporates the kinematic constraints of the specified robot, such as linear and angular acceleration limits, to produce an optimal, viable pathway in a given localized window. The output is a circular path, defined by translational and angular velocity.

The DWA algorithm works by projecting a set of sampled velocities across the window, and evaluating the score of each trajectory as dictated by the objective function. The highest scoring trajectory is selected, and then the cycle repeats. The limited search space confined by the dynamic window allows a solution to be computed. These individual trajectories stitch together to follow the global path.

3.3.3 Timed Elastic Band

The Time Elastic Band (TEB) approach is another common implementation of the local path planner. Naive Elastic band planners, first proposed in 2002 [32], produce a local route by deforming the global pathway in real to time avoid obstacles. Much like a rubber band, the pathway produced simply bends around local obstacles. Timed elastic bands, first proposed in 2012 [33], expand on this concept by also incorporating the kinematic constraints of the robot.

3.4 Conclusion

The algorithms necessary to plan and execute trajectories through mapped space have been presented. The well established path planning algorithms of A^* and RRT^*, and tra-

jectory planning algorithms of *DWA* and *TEB* serve as the core for most Autonomous navigation today, and can even be sufficient in their naive form for some systems. Current research in the field, specifically in the realm of self driving cars, addresses challenges of avoiding dynamic objects, such as other cars or pedestrians, with extremely high precision.

CHAPTER 4

ROS and Gazebo

The open source robotics middle-ware Robot Operating System (ROS) from Open Robotics [31] was used to implement SLAM algorithms on a real-world robot using LIDAR and wheel odometry. The SLAM packages of Gmapping[19], and Cartographer [21] were implemented.

4.1 ROS structure

ROS is an open source software framework to aid in the development of complex robotic systems. Much like an operating system, it provides a high level abstraction for low-level hardware control which allows for rapid, reliable development. It is language-neutral, with most code written in C++ or Python. The open source nature of ROS provides a large database of software packages to choose from, making the integration of available code trivial. Development of complex robotic systems without such an open source framework are beyond the capability of any single developer.

At a high-level, ROS consists of peer-to-peer nodes that are responsible for performing specific computation. Nodes receive input by subscribing to a desired topic, and produce output by publishing under a specified topic. Together, the communication between nodes define the functionality of the robot. The modularity of this structure allows the user to easily add or remove nodes from the robot, as well as making it robust to hardware failure.

4.2 SLAM packages

SLAM is an active area of research, with many ROS packages available. Some of these SLAM packages include (a) GMapping: A laser-based SLAM package developed by the OpenSlam foundation. It employs a Rao-Blackwellized particle filter (RBPF) for 2-D occupancy grid mapping [19] with a laser scanner and odometry data; (b) Hector SLAM: A laser-based SLAM package developed by the Technische Universitat Darmstadt. It employs scan matching to build a 2-D occupancy grid [25] from a laser scanner and a 3-D navigation system such as an IMU; (c) Cartographer: A robust Graph-based SLAM package developed by Google. It is a highly flexible platform to accommodate a wide range of sensor configurations [21] and algorithms; (d) SLAM Toolbox: A robust Graph-based SLAM package developed by Steve Macenski. It employs graph-optimization techniques (Google Ceres [3]) to provide accurate 2-D occupancy grid mapping capabilities through laser scanners and odometry data [27]. It is the default SLAM toolbox for ROS 2.

4.3 ROS and Gazebo

A common robot development process is to create a model of the hardware within a 3-D physics based robotics simulation to emulate the robot motion. By using such a framework, the software of the robot can be easily tested on the virtual hardware, allowing rapid iteration of both the hardware and controller. Within computational resources, it is ideal for the simulation to be as close as possible to the real world to minimize the gap between simulation and real-world hardware. There are many simulation software packages to choose from such as CoppeliaSim, Webots, V-REP, RobotDK, and Gazebo. Gazebo is the chosen software for this project due to it's native integration with ROS.

4.3.1 Overview of Gazebo

Gazebo is an open source robotics simulator maintained by Open Robotics, and first published in 2005 [24]. It consists of three main modules: (1) A physics engine capable of simulating rigid body dynamics. Gazebo can interface with a variety of engines such as Bullet, Simbody, DART, and ODE. ODE is the default engine. (2) A visualization engine capable of rendering 3-D graphics. OpenGL and GLUT are the default graphics engine.(3) An environment made up of rigid body models, joints, sensors, and an external interface for control. As Gazebo is the native simulator for ROS, ROS provides the external interface. From the viewpoint of ROS, there is no difference between the actual hardware and the virtual hardware simulated in Gazebo. Although there is no substitute for evaluating real hardware performance, this critical feature is invaluable when developing software packages.

The model contains the core functionality of the physical hardware. This includes a high-fidelity model of the Velodyne VLP-16 LIDAR, simple wheel odometry, and a simple transmission for the differential drive package. The parameters of wheel size, wheel placement, overall weight, and LIDAR placement were carefully matched to the real world robot to maximize similarity of system performance. Shown in Figure XXX is the virtual environment used to test the robot performance. It was constructed to match the degree of difficulty to the real world in lab and around campus.

4.4 ROS packages

The relevant ROS packages used in this project are tabulated below.

ROS Packages			
ROS package	Description	Published Topics	Subscribed Topics
diff_drive_controller	Differential drive controller for publishing motor commands	odom, tf, publish_cmd	cmd_vel
tf	Transform package to maintain relationship between coordinate frames	/tf_changes	/tf
joy	Driver for joystick for teleoperation	joy	N/A
move_base	Implementation of the Action Library to move the mobile base towards set goal	cmd_vel	move_base_simple/goal
amcl	Probabilistic localization with adaptive Monte-Carlo localization	amcl_pose, particlecloud, tf	scan, tf, initialpose, map
gmapping	Package for particle filter, laser based SLAM	map_metadata, map, entropy	tf, scan
Slam$_{t}$*oolboc*	Package for Graph-based SLAM	TBD	TBD
amcl	Adaptive Monte Carlo Localization package	amcl_pose, particlecloud, tf	scan, tf, initalpose, map
base_local_planner	Implementation of trajectory rollout approaches for local navigation around obstacles	global_plan, local_plan, cost_cloud	odom
dwa_local_planner	Dynamic Window approach plugin for base_local_planner	global_plan, local_plan	odom
teb_local_planner	Time Elastic Band planner plugin for base_local_planner	global_plan, local_plan, teb_poses teb_markers, teb_feedback	odom, obstacles, via_points

4.4.1 Navigation Stack package

The navigation stack is a complex set of ROS packages that take sensor streams, the current pose, and the goal pose to produce velocity commands for the mobile base. Shown below is a depiction of the stack. Light blue boxes show data sources specific to each individual robot. For example, the sensors can be from a laser scan produced by LIDAR, or a Point Cloud produced from LIDAR or a camera. The light grey boxes are optional packages that can be used within the stack. The white circles are compulsory ROS packages that define the behavior of the stack. For example, the global_costmap takes in sensor data to produce the current map, which is then fed into the global planner. The output from the global planner feeds into the local_planner, which also takes in the local_costmap, and ultimately produces velocity commands for the base_controller to execute.

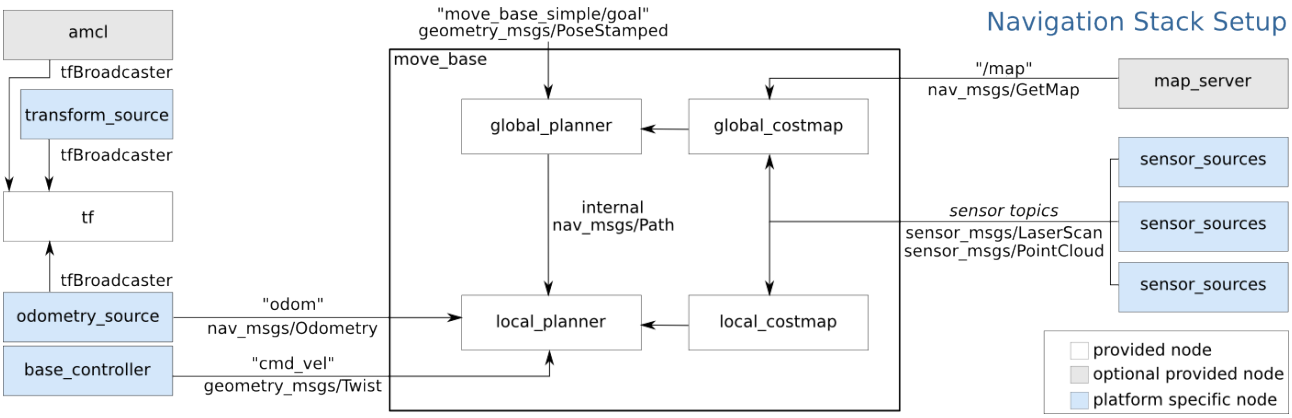

Figure 4.1: Depiction of navigation stack [2]

4.4.2 AMCL package

When a map is provided, AMCL is responsible for localization within that map. It utilizes a particle filter to estimate the pose of the robot within the known map. As noted in the above table, it takes in an occupancy grid, laser scans, transform messages, and computes pose estimates based off the motion and laser models described in previous sections.

4.4.3 tf package

The tf package [11]is responsible for tracking all reference frames, such as the base_frame, laser_frame, odometry_frame, etc. and computing transforms between them. It is powerful, critical package for nearly all robotic systems built with ROS. Shown below is a depiction of the coordinate frame transformation as computed by the tf package. At the top of the transform tree is the map frame, which broadcasts it's coordinate frame to the odometry frame, and finally to the base_link frame. As expected, the base_link frame is the parent frame to most of the frames within the system. It should be noted that each frame has a specified tolerance in second of allowing required frames to lag behind. Critical packages like the navigation stack will stop sending velocity commands in

the frame transformation between the map and base_link takes longer than the specified tolerance to prevent unwanted behavior.

4.5 Robot Description

A robot description file, typically in the form of a Unified Robot Description Format (URDF) is required for both the real world robot and the Gazebo Model. The Gazebo model should take the same URDF, and simply add the necessary ROS-Gazebo plugins and visual/inertial elements for simulation. The URDF is responsible for specifying the necessary parameters used in other ROS packages. This includes specification of the wheel radius and left and right wheel distance for the odometry motion model, and location of specified coordinate frames such as the laser mount relative to the base frame.

4.6 Tuning the navigation stack

Setting up the global and local path planner for a specified robot is a notoriously difficult task. An excellent starting point is Zheng's navigation tuning guide for ROS [42]. Publicly available code for educational mobile robots such as Robotis's TurtleBot3 or Clearpath Robotic's Husky are also invaluable starting points, but need to be modified for each individual robot. In particular, robust odometry, accurate coordinate transformations, and computationally tractable global and local planner parameters are critical to successful navigation.

Robust odometry was achieved by carefully adjusting the wheel radius and separation between the left and right wheel until satisfaction. This was done by setting a long decay time of 15 seconds on the laser scan in rviz, and looking for smearing in the laser scan visualisation when under translational or angular velocity. Ideally the laser scan reading should continue to overlap under motion. If unreasonable smearing was

found, the odometry motion model was adjusted until the laser scan remained roughly overlapped with itself. This same method was used to find the correct location for the laser scan transform.

For trajectory planning, *TEB* was chosen over *DWA* do to its superior performance in efficiency, safety, and flexibility [39]. The *TEB* local planner was tuned to minimize backwards driving, incentive straight line movement and in place turning, and be able to fit through doorways. This was achieved by adjusting tuning the parameters *weight_kinematics_forward_drive*, *min_obstacle_distance*, and the minimum and maximum velocities.

4.6.1 Cost Maps

Costmaps are a ubiquitous concept across all robotic systems. They are a simple way to encode cost of traversing a particular section of the configuration space. In the context of navigation, specifically 2-dimensional navigation within occupancy grid maps, cost maps store and update information about obstacles in the grid map. The costmaps are used by the global and local path planners to find optimal pathways. The costmaps are a critical piece to the overall navigation problem, and must be tuned accordingly.

In ROS, the costmap_2d package uses a set of layers to define the costmap. A static, obstacle, and inflation layer typically make up the overall costmap.

4.6.1.1 Static Layer

The static layer typically takes a previously generated map from SLAM, and adjusts the cost of the mapped objects to the desired values. In particular, it is important for the user to define the *unknown_cost_value* which specifies if an unknown area is considered free or occupied, and the *lethal_cost_threshold* which defined the value at which a cost is considered lethal for traversing. Setting the lethal cost to a reasonable value is necessary

for good global path planning.

4.6.1.2 Obstacle Layer

The obstacle layer incorporates obstacles as seen in real time as measured by the sensor suite. This is typically performed with the same system that executed SLAM, in this case LiDAR. It sits on top of the static layer. The obstacles incorporated can be represented as either 2-dimensional additions, or 2-dimensional additions to the costmap

It is responsible for both identifying new objects, and removing objects according to some threshold. This process is referred to as marking and clearing of obstacles. Marking is the basic process of identifying new objects with the laser scan, and clearing is a more complex process of performing ray tracing through the object to see if it has disappeared. Special care must be taken to tune the parameters specific to cl earning and removing obstacles to see reasonable results.

4.6.1.3 Inflation Layer

The inflation layer is an additional layer that simply inflates obstacles to neighboring cells. This is an easy way to force the costmap to account for the configuration space of the robot when performing global path planning such as A^*. It is important to tune the inflation *inflation_radius* and *cost_scaling_factor* to the specifics of the robot to reasonable trajectories.

CHAPTER 5

Data Collection and Analysis

The two SLAM algorithms of Gmapping and Cartographer were implemented and compared. The robot was teleoperated around campus with an Xbox One controller.

5.1 Hardware

The hardware for Antbot is shown in the Table below

Antbot Hardware list	
Component	
Computing module	Nvidia Jetson TX2
Storage	Samsung 850 EVO 250Gb
Lidar	Velodyne Lidar VLP-16
Motor Controller	RoboteQ SDC2130
Motors	IG52-04 24VDC with encoder
Controller	Xbox One Controller
Chassis	Superdroid IG52-DB4 All Terrain Platform

5.2 SLAM

Gmapping and Cartographer were implemented to map the surrounding environment. Although a similar quality of map is obtained, Cartographer is significantly more robust and capable. It is able to produce a globally consistent map through robust Loop Clo-

sure, and continually update the map through life long mapping modes made possible through the Graph-based SLAM framework.

5.2.1 Gmapping

GMapping is a laser based SLAM package that employs a Rao-Blackwellized particle filter (RBPF) for 2-D occupancy grid mapping. As an online filtering method, it suffers from large mapping problems. Shown below is the map gathered from using GMapping while teleoperating in a clockwise direction around the fourth floor of Engineering Building IV.

As a featured based mapping algorithm, producing a map through relatively featureless corridors can be quite challenging. Nonetheless, the quality of the map produced is quite accurate. It successfully maps all of the 90 degree corridor turns and produces consistent walls for nearly the entirety of the map. The most interesting area is the point at which the robot makes a complete loop and enters the same area in which it has already seen. This is the far left portion of the map, where the cavity mapped in the SMERC lab. Inconsistency in the map between occupied and unoccupied territory can been seen in the free space appearing to the left of the already mapped wall. The system does doe an excellent job of locating within the doorway, producing consistent map as the critical point. Overall, this map is accurate enough for autonomous navigation.

5.2.2 Cartographer

Cartographer (not shown) is sensor agnostic graph-based SLAM package for online and offline 2-dimensional and 3-dimensional mapping. As an offline optimization method, it is capable of mapping very large spaces, and to continue refine a map through continual loop closures. It arguably the most general purpose and feature-rich open-source SLAM package currently available.

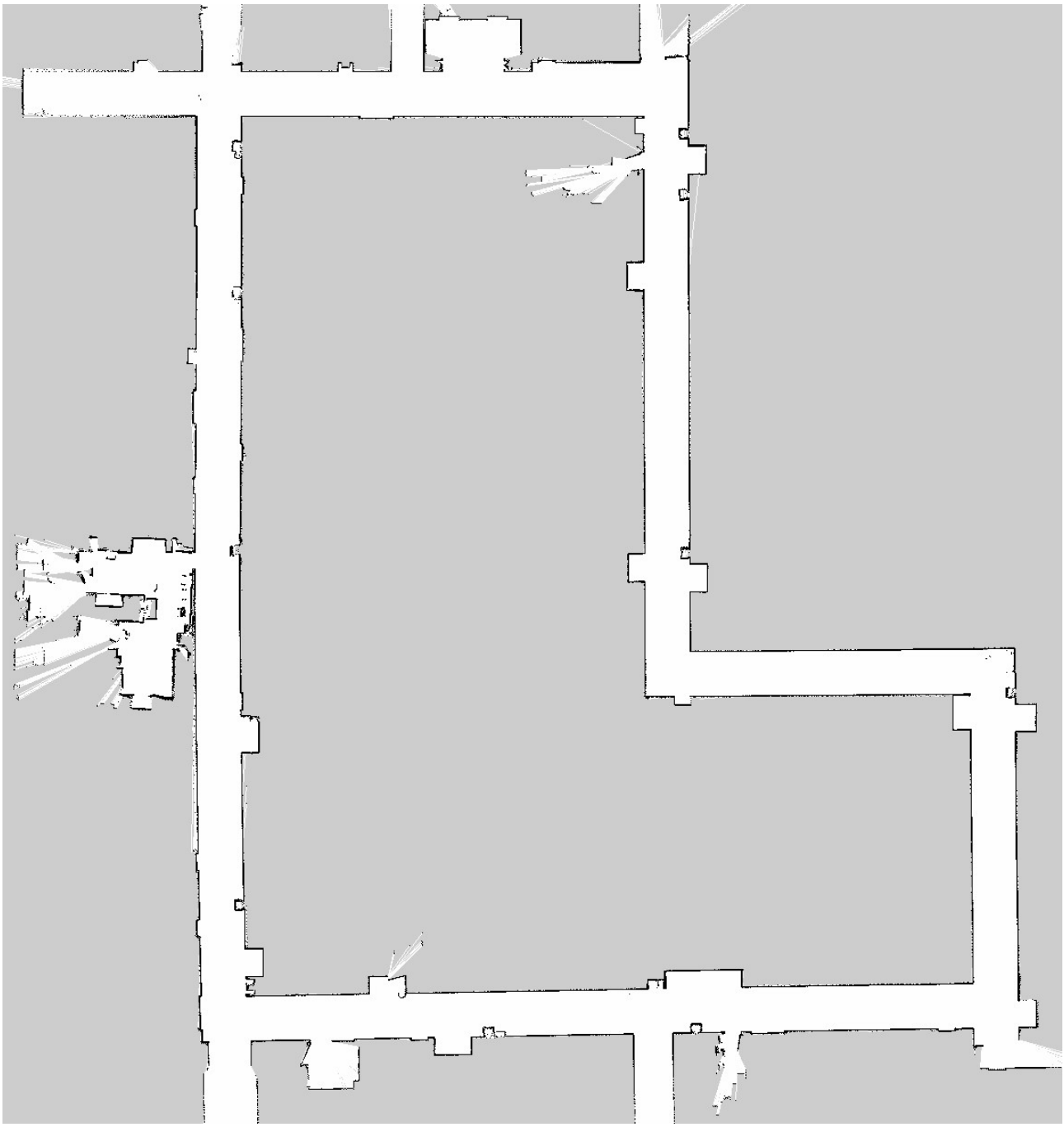

Figure 5.1: Map produced from GMapping

5.3 Navigation

After obtaining a map of the environment, *AMCL* was used for localization. Shown below are two sequential images showing the initial particle filter estimate for the pose

$\mathbf{x_t}$, and the converging estimate after moving a couple meters forward.

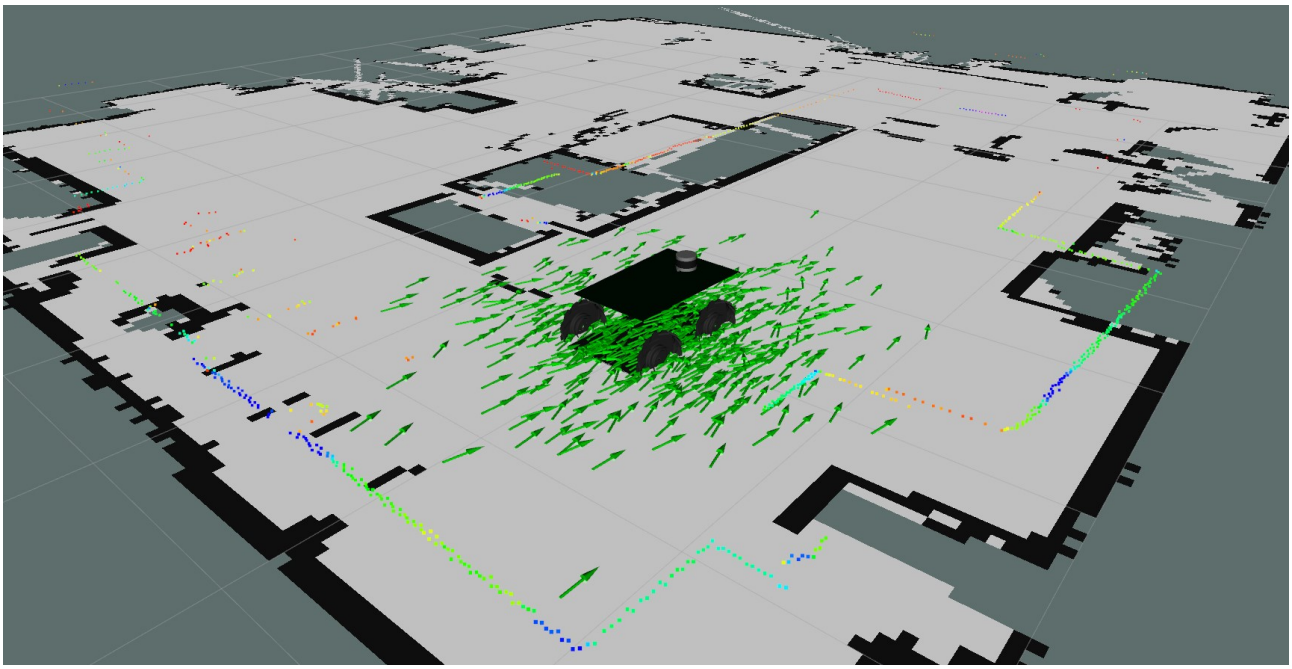

Figure 5.2: AMCL initial particle distribution

To start navigating autonomously, rviz was used to specify a desired goal locations for the robot to drive towards. Shown below is a printout of the rviz screen during navigation.

The rendered robot in rviz depicts the actual pose of the robot, and the goal is shown as the large red arrow. The global path as computed by the A^* algorithm is shown in red. It uses the global costmap to find an optimal trajectory for the point-mass robot. The local path as computed by TEB is shown in yellow. It samples trajectories from the motion model, scores them as defined by the local costmap, and then chooses the best trajectory within the window. It is capable of dynamically re-planning in the presence of obstacles, and is ultimately responsible for sending control commands to the motors. The global costmap is shown as the black shaded outline around obstacles, and the local costmap is shown as the colorful map in the local window. Areas of high intensity are more costly, resulting in the global and local path planner sticking to the cheaper middle

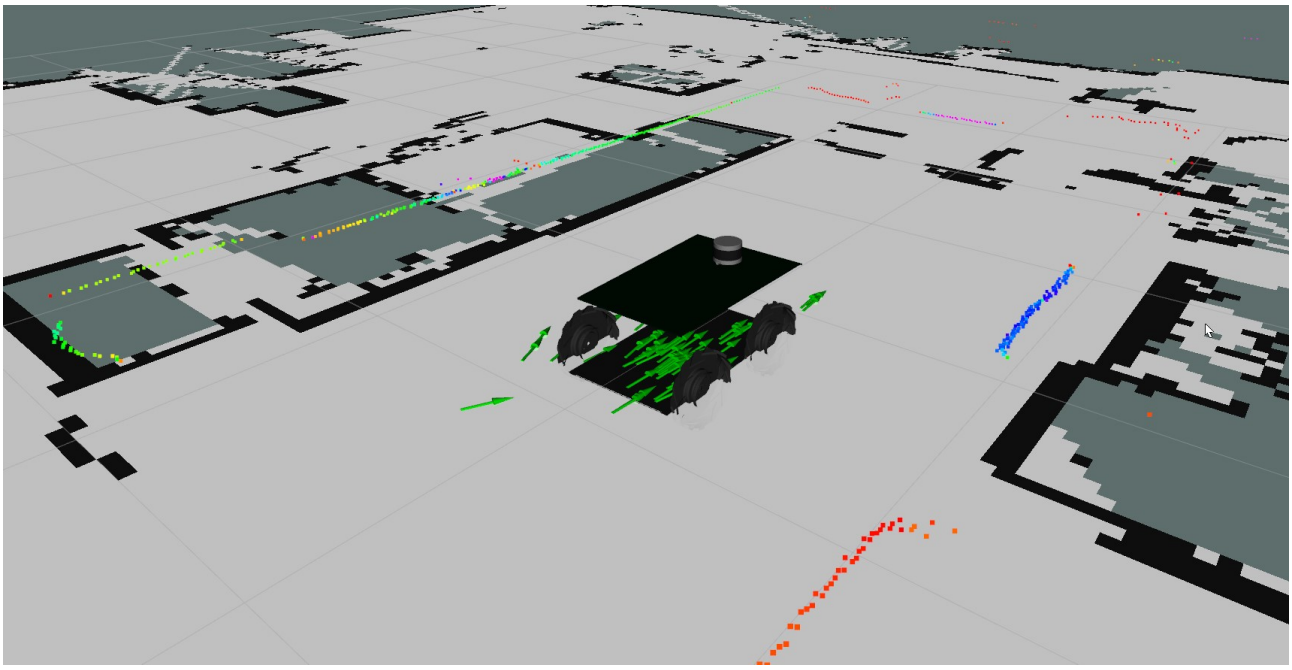

Figure 5.3: AMCL updated particle distribution after driving

section as desired. The costmap was specifically tuned to generate this behavior.

Overall, the global path planner performs quite well, and is able to find a route to the desired location with little to no difficulty. The local path planner was much more difficult to tune, and can struggle to find the optimal pathway. Many kinks such as sub-optimal paths had to be ironed out to see successful navigation.

Shown below is the navigation pathway through the doorway of SMERC lab. As the robot is only a few inches narrower than the doorway itself, the local planner had to be carefully tuned to allow passage, and enforce a centered pathway.

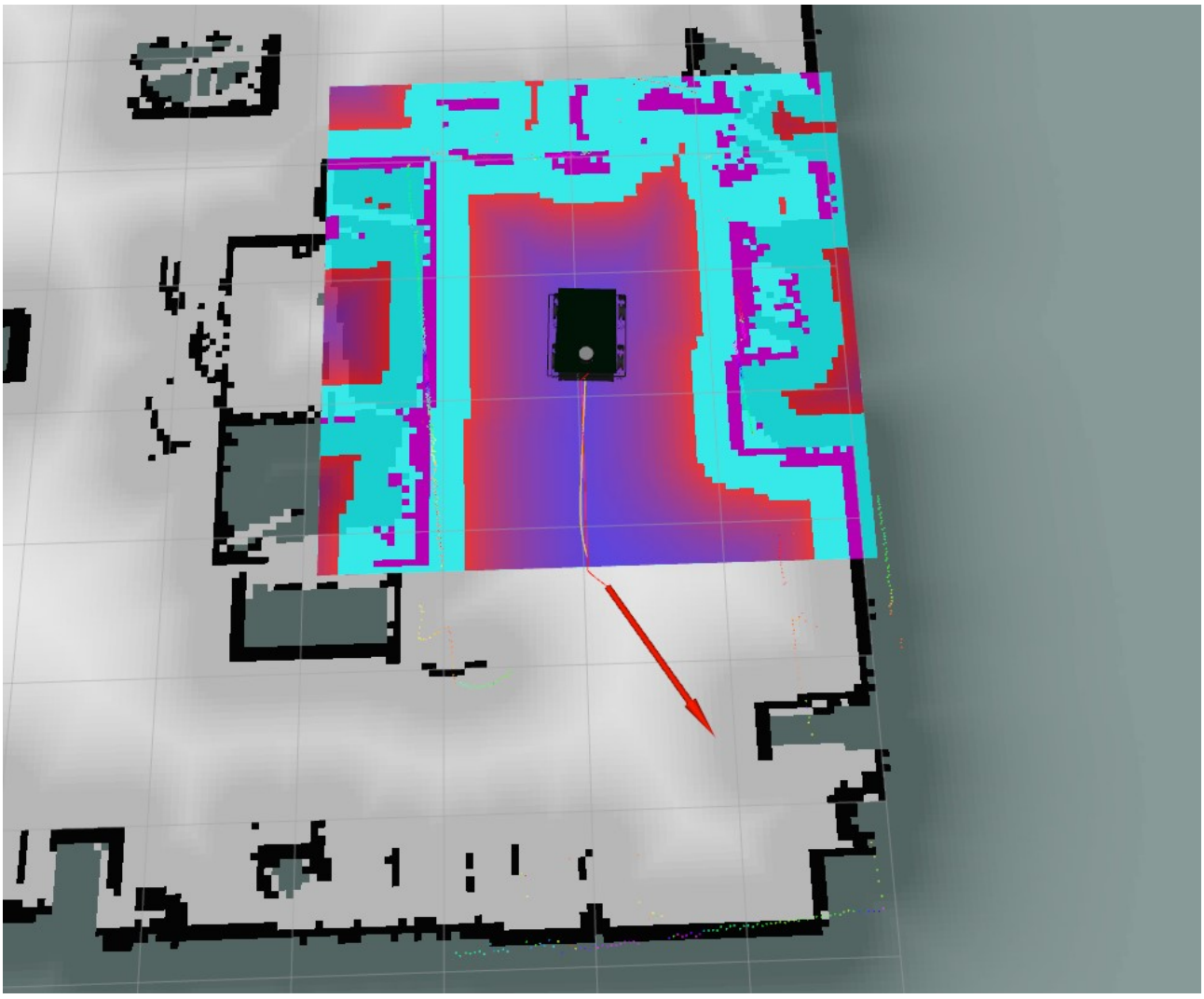

Figure 5.4: Navigation in rviz

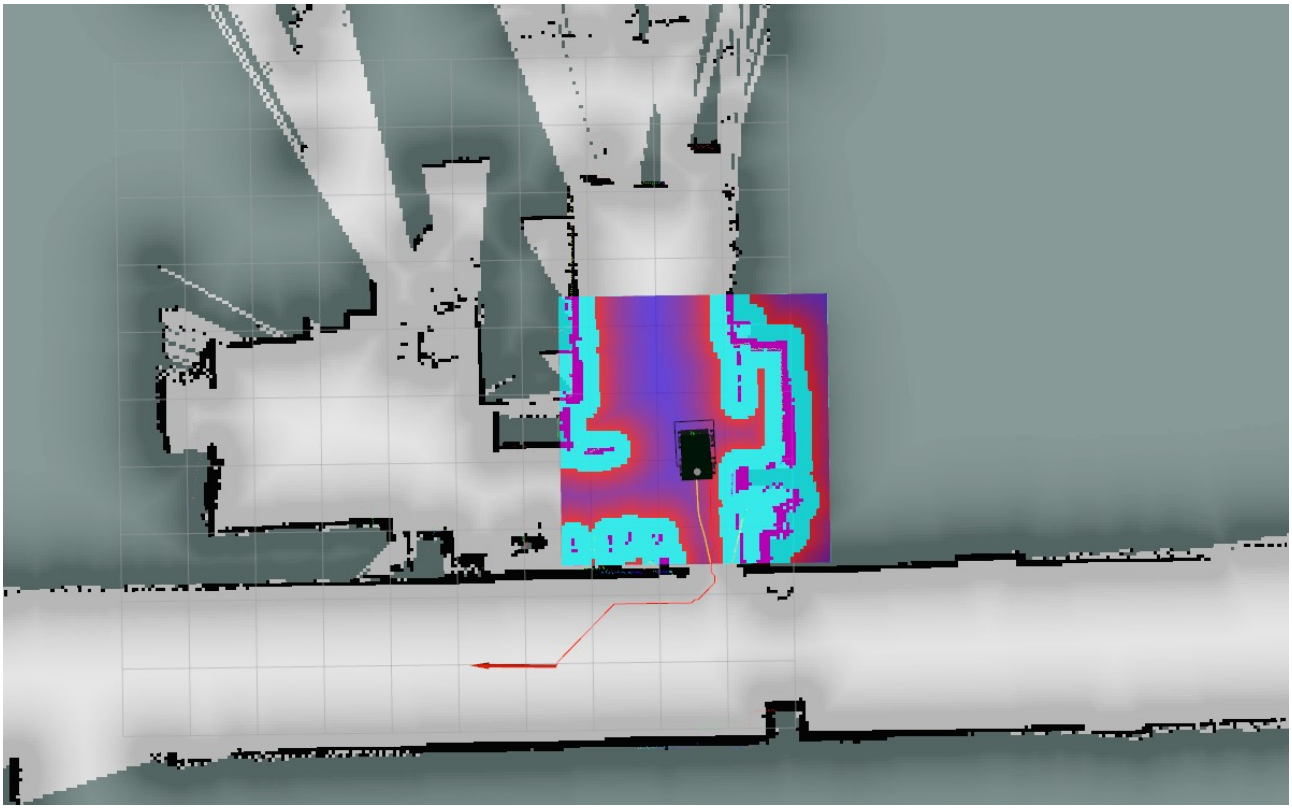

Figure 5.5: Navigation through doorway

CHAPTER 6

Conclusion of work

This project has provided an introductory overview of autonomous navigation. SLAM provides a pragmatic solution to well defined localization and mapping problems. And, shortest path algorithms in conjunction with trajectory planning techniques provide a reasonable framework for motion planning and control within the mapped environment. Implementation of these algorithms was demonstrated with a real world robot, capable of mapping and navigating autonomously. It was also shown how this relatively naive approach of constructing an occupancy grid map, and navigating through that map, quickly becomes impractical for large real world problems such as self driving cars. Although successful deployments of SLAM can be found in iRobot Roombas, Skydio drones, Tesla vehicles, and even Mars rovers Opportunity and Perseverance, the overarching goal of truly robust and long term autonomy is unsolved. Recent advancements in SLAM, specifically in real world products from companies such as Tesla and Skydio, have demonstrated the dominance of visual based methods.

6.1 Future Steps

One of the primary challenges to truly autonomous robotic systems is engineering a robot capable of resolving failure without human intervention. Although recent developments in powerful open-source SLAM packages like Cartographer and SLAM Toolbox [27] are capable of performing lifelong mapping, and handling uncertain failure modes

such as the kidnapped robot scenario [35], there is still much work to be done. Robust perception, the ability to perceive the environment across difficult circumstances is an open research field. Dusty environments, translucent objects, camera saturation, sensor failure, and unforeseen events remain challenges for even state of the art SLAM systems. Current trends in SLAM are building on the current foundation by solving the semantic mapping problem. Semantic SLAM aims to provide an additional layer of class information on mapped objects. Similar to how humans reason about their own environment, this additional layer of abstraction promises to provide efficient and robust data structures for path planning. At an even broader level, it can be noted that the entirety of SLAM research is converging towards the broader goal of general spatial AI [6]. Spatial AI is a broad field research that aims to perceive, and reason about visual imagery in an intelligent way. There is currently debate about the necessity of SLAM, and questions on how much Machine Learning and Computer Vision will replace classically formulated SLAM architecture. Intuitively, it is difficult to imagine our own brains employing something similar to the rigid constraints of a system like Graph-based SLAM. We will likely see a wider adoption of AI driven techniques to solve both SLAM and Motion Planning over the coming years.

REFERENCES

[1] base_local_planner description of base_local_planner package for trajectory rollout and dwa approaches. http://wiki.ros.org/base_local_planner. Accessed: 2021-09-21.

[2] move_base description of move_base package for mobile navigation. http://wiki.ros.org/move$_b$*ase*?*action* = *AttachFiledo* = *viewtarget* = *overview.png*.

[3] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org.

[4] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[5] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J.J. Leonard. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[6] Andrew J. Davison. Futuremapping: The computational structure of spatial ai systems, 2018.

[7] Frank Dellaert and Michael Kaess. 2017.

[8] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[9] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks, 2000.

[10] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[11] Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6, April 2013.

[12] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, 4(1):23–33, 1997.

[13] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep 2014.

[14] Simon Godsill. Particle filtering: the first 25 years and beyond. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7760–7764, 2019.

[15] H.H. Goldstine and A. Goldstine. The electronic numerical integrator and computer (eniac). *IEEE Annals of the History of Computing*, 18(1):10–16, 1996.

[16] N. Gordon, D. Salmond, and Adrian F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. 1993.

[17] Mohinder Grewal and Angus Andrews. Applications of kalman filtering in aerospace 1960 to the present [historical perspectives. *Control Systems, IEEE*, 30:69 – 78, 07 2010.

[18] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[19] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[20] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[21] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.

[22] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[23] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning, 2011.

[24] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[25] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[26] S. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.

[27] Steve Macenski and Ivona Jambrecic. Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, 6(61):2783, 2021.

[28] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. 11 2002.

[29] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 116–121, 1985.

[30] P. Ogren and N.E. Leonard. A convergent dynamic window approach to obstacle avoidance. *IEEE Transactions on Robotics*, 21(2):188–195, 2005.

[31] M. Quigley. Ros: an open-source robot operating system. In *ICRA 2009*, 2009.

[32] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2, 1993.

[33] Christoph Rösmann, W. Feiten, Thomas Wösch, F. Hoffmann, and T. Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *ROBOTIK*, 2012.

[34] Anthony (Tony) Stentz. The D* algorithm for real-time planning of optimal traverses. Technical Report CMU-RI-TR-94-37, Carnegie Mellon University, Pittsburgh, PA, October 1994.

[35] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[36] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *I. J. Robotic Res.*, 25:403–429, 05 2006.

[37] C.H. Leng M. Yogeswaran O.E. Ng Y.Z. Chong T.J. Chong, X.J. Tang. Sensor technologies and simultaneous localization and mapping (slam). *Procedia Computer Science*, 76:174–179, 2015.

[38] Eric Wan. Sigma-point filters: An overview with applications to integrated navigation and vision assisted control. In *2006 IEEE Nonlinear Statistical Signal Processing Workshop*, pages 201–202, 2006.

[39] Jian Wen, Xuebo Zhang, Qingchen Bi, Zhangchao Pan, Yanghe Feng, Jing Yuan, and Yongchun Fang. Mrpb 1.0: A unified benchmark for the evaluation of mobile robot local planning approaches. 11 2020.

[40] Kai M. Wurm, Cyrill Stachniss, and Giorgio Grisetti. Bridging the gap between feature- and grid-based slam. *Robotics and Autonomous Systems*, 58(2):140–148, 2010. Selected papers from the 2007 European Conference on Mobile Robots (ECMR '07).

[41] C. Zammit and E. Kampen. Comparison between A* and rrt algorithms for uav path planning. 2018.

[42] Kaiyu Zheng. *ROS Navigation Tuning Guide*. 06 2017.