PATH PLANNING FOR ROBOT NAVIGATION IN SPARSELY OBSERVABLE ENVIRONMENTS

Kent H. Altobelli
University of Rhode Island, kentaltobelli@gmail.com

PATH PLANNING FOR ROBOT NAVIGATION IN SPARSELY OBSERVABLE ENVIRONMENTS

BY

KENT H. ALTOBELLI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2023

MASTER OF SCIENCE THESIS

OF

KENT H. ALTOBELLI

APPROVED:

Thesis Committee:

Major Professor	Paolo Stegagno
	Kaushallya Adhikari
	Chengzhi Yuan
	Brenton DeBoef
	DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

2023

ABSTRACT

A path planning method using Model Predictive Control (MPC) is explored for driving the state of the robot towards a goal while utilizing knowledge of which states are fully observable to simultaneously minimize state uncertainty. State estimation is accomplished using a particle filter. Path planning is accomplished using multiple particle filters to track the performance of the hypothetical state estimates while propagated out to a specified time horizon. Paths are continuously evaluated using a scoring function, yielding a final "best path" selection that is implemented by the robot controller to achieve the best blend of performance - seeking the goal while simultaneously minimizing state estimate uncertainty. Crucially, hypothetical paths are generated using breadth-first search of the state space and not intentionally driven towards observable states. The proposed path planner was tested in simulation and gave superior results over a naive navigation scheme in many evaluated scenarios of sparse environment observability.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to Dr. Paolo Stegagno, my major professor and chair of the committee, for countless hours of instruction and access to the Intelligent Control and Robotics Laboratory equipment for simulation validation.

I am also grateful to my partner, Elizabeth, for continuous encouragement and support which allowed me to focus on this research.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table **Page**

CHAPTER 1

Introduction

For mobile ground robot state estimation, using proprioceptive sensors for dead reckoning is subject to unbounded drift over time due to the accumulation of error, so an exteroceptive measurement source is often used to correct state estimates. However, some systems may not be capable of gathering sufficient measurements to determine an accurate state estimate due to the unavailability of measurement sources, increased system cost/complexity to access those measurement sources, or an environmental limitation (e.g. a vision based system operating in a pitch black room). In these cases there is a need for robots to operate long enough to reach a desired goal state, or reach an area where the system can successfully obtain measurements en route to the goal.

In this research, a solution for path planning is explored to minimize state estimate uncertainty over the course of a mission to a goal state despite requiring the system to operate periodically in regions that cannot guarantee exteroceptive measurements. Like most systems, the robot used for this research will be capable of operating in fully observable areas of the state space, but critically, the environment will be constructed in a way that requires the robot to navigate across arbitrary gaps in observability to accomplish mission objectives. During these short periods of time, proprioceptive sensors will continue to gather information for state estimates, but error is expected to increase until the system finds a fully observable state to reconverge state estimates. This approach will effectively trade continuous certainty of state variables for increased flexibility and range beyond typical operation points.

1.1 Background

1.1.1 Particle Filter

One challenge at the heart of every control system problem is developing an accurate state estimate to determine the control action that will drive the system to the goal. In motion planning, this state estimate has the additional affect of potentially setting the start point for the path in the wrong location compared to the ground truth. Authors in [1] provide a comparison of various approaches to robot localization. For this research we are specifically interested in the performance of Bayesian filter derivatives commonly used in robot localization like the Kalman filter, grid-based Markov localization, and sampling-based methods.

For completeness, a derivation of the Bayes Filter is included here following the procedure from [2]. The notation for this derivation requires a series of states $\{x_0, x_1, ..., x_t\}$ which can be expressed as $x_{0:t}$, a series of inputs $\{u_1, u_2, ..., u_t\}$ which can be expressed as $u_{1:t}$, and a series of measurements $\{z_1, z_2, ..., z_t\}$ that can be expressed as $z_{1:t}$. We assume a known probability distribution of the initial state $p(x_0)$, a known measurement probability $p(z_t|x_t)$ for each time t, and a known state transition probability $p(x_t|x_{t-1}, u_t)$.

We would like to find the posterior distribution belief $bel(x_t)$ at time step t as the conditional probability of our current state x_t given the all previous measurements $z_{1:t-1}$ and all previous control inputs $u_{1:t}$:

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

given the posterior distribution from the previous time step $t - 1$:

$$bel(x_{t-1}) = p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) \tag{2}$$

Recall Bayes' theorem to compute the conditional posterior probability of a given b and c:

$$p(a|b,c) = \frac{p(b|a,c)p(a|c)}{p(b|c)} \tag{3}$$

To express the desired posterior distribution in Equation 1, we split the measurements $z_{1:t}$ into the most recent measurement z_t and previous measurements $z_{1:t-1}$ and define the events where $a = x_t$, $b = z_t$, and $c = x_0, z_{1:t-1}, u_{1:t}$.

Substituting into Bayes' theorem into Equation 3 yields:

$$p(x_t|z_t, z_{1:t-1}, u_{1:t}) = \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \tag{4}$$

where the denominator $p(z_t|z_{1:t-1}, u_{1:t})$ is a scalar value representing a normalization factor η to ensure that the integral of the posterior distribution over all conditionals goes to 1. The measurement notation for all measurements $z_{1:t}$ is also collapsed in the posterior belief for simplicity.

$$p(x_t|z_{1:t}, u_{1:t}) = \eta p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t}) \tag{5}$$

The Bayes filter makes a crucial assumption that evolution of the system represents an unobserved Markov process, making the current state x_t a complete representation of all previous measurements $z_{1:t-1}$ and inputs $u_{1:t}$. Therefore Equation 5 can be further simplified to:

$$p(x_t|z_{1:t}, u_{1:t}) = \eta p(z_t|x_t)p(x_t|z_{1:t-1}, u_{1:t}) \tag{6}$$

We can now define an "a priori" belief of the distribution of state x_t before we receive the current measurement as $\overline{bel(x_t)} = p(x_t|z_{1:t-1}, u_{1:t})$, allowing us to make a final substitution to yield a standard form of the measurement update portion of Bayes' filter algorithm.

$$bel(x_t) = \eta p(z_t|x_t)\overline{bel(x_t)} \tag{7}$$

To investigate the implications of the $\overline{bel(x_t)}$ term, recall the total law of probability:

$$p(a) = \int p(a|b)p(b)\,d(b) \tag{8}$$

where $a = x_t|z_{1:t-1}, u_{1:t}$ and $b = x_{t-1}$, which yields the total probability of the the a priori distribution from a distribution conditioned on the previous state x_{t-1} and knowledge of the probability distribution of that state.

$$\overline{bel(x_t)} = \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t})p(x_{t-1}|z_{1:t-1}, u_{1:t})\,dx_{t-1} \tag{9}$$

Once again, applying the assumption that the current state x_t is a complete representation of previous measurement and inputs, we simplify the conditions $x_{t-1}, z_{1:t-1}, u_{1:t}$ to x_{t-1}, u_t. Finally, the input u_t has not yes been commanded for state x_{t-1} so that element is removed from the conditional $p(x_{t-1}|z_{1:t-1}, u_{1:t})$ to yield the common form of the a prior belief of state x_k computed for the state transition probability after receiving an input command u_t, but before measurement z_t:

$$\overline{bel(x_t)} = \int p(x_t|x_{t-1}, u_t)p(x_{t-1}|z_{1:t-1}, u_{1:t-1})\,dx_{t-1} \tag{10}$$

The Kalman filter proposed in [3] implements the Bayes Filter with the assumption of Gaussian distributions for the posterior probability distribution of the state, a priori probability distribution of the state transition outcome, and probability distribution of measurement model. Fundamentally, this makes the Kalman filter a practical application of the Bayes Filter because results can be calculated in closed form due to the additive and multiplicative properties of Gaussian distributions yielding a Gaussian-distributed result.

For nonlinear problems however, Monte Carlo sampling-based methods proposed in [4] and [5] can be used to approximate the probability distribution functions discussed in the Bayes Filter. Various versions the particle filter have emerged over time, but the basic assumptions and methods are still derived from Sequential Monte Carlo algorithms as summarized by the authors of [6]. The particle filter uses many weighted particles to approximate the probability distributions presented in the Bayes filter, yielding probability mass functions as the discrete equivalent. Given a set of particles forming a probability mass function representing the previous system state posterior belief $bel(x_{t-1})$, the particle filter iteratively approximates the updated state probability distributions in two distinct phases:

- Given an input u_t, the a priori probability distribution $\overline{bel(x_t)}$ is approximated by applying the state transition model to each particle with additional noise sampled from an arbitrary probability distribution function representing the state transition uncertainty. This "time update" of the system model yields a probability mass function approximating the Bayes filter's a priori belief described by Equation 10.

- When a measurement z_t is received, the new posterior probability distribution $bel(x_t)$ is approximated by multiplying the weight of each particle by the probability distribution function value at each particle's expected measurement value for an arbitrary probability distribution representing the measurement uncertainty. Finally, the weights of all the particles are normalized to sum to 1, restoring the weights of the particles to form a legitimate probability mass function. This "measurement update" using the measurement model yields a probability mass function approximating the Bayes filter's new posterior belief described by Equation 7.

One challenge with the implementation of a standard particle filter is particle degeneracy as discussed in [7], where the weights of some particles approach zero due to the evolution of some particles toward states with a low likelihood of representing the state probability distribution function. The particle filter is unable to glean much information from these degenerate particles, so with a fixed number of particles employed by the particle filter, the number of useful particles contributing to system state estimation may become significantly reduced over time, giving poor long term performance. A common solution to this problem is an unbiased particle Importance Resampling (IR) where particles are resampled to form a proposal distribution identical to the original state probability distribution, but other methods exist for achieving the desired proposal distribution as discussed in [8].

1.1.2 Observability

When designing a state estimator, the topic of observability must first be addressed, namely that the measurements of a system are sufficient to uniquely reconstruct the state of that system [9], [10]. An intuitive way to define observability is presented by the authors of [11], where a system is considered to have states x belonging to the state space M. An "input-output map" is formed for a system with initial state x_0 given all possible inputs over a time interval $[t^{(0)}, t^{(1)}]$ and measuring the resulting system output. If the same system with a different initial state x_1 yields the same input-output map over the same time interval $[t^{(0)}, t^{(1)}]$, then the evolution of the two states is indistinguishable. Therefore, a system is only said to be observable at an arbitrary state x if there are no other initial states result in the same input-output map. Furthermore, the system as a whole is observable if every system state x, $\forall x \in \{M\}$ is observable.

An observability study of mobile ground robots with unicycle dynamics (the same used in this research) is conducted by the authors of [12]. Their work tabulates the observability of a two robot system with various combinations of control inputs and measurements, so for applicability to this research, the second robot is assumed to be a stationary beacon by setting the velocity $v = 0$. Therefore, the state of our robot is found to be fully observable if the robot is moving with velocity $v > 0$ and the robot is measured at a bearing of α from the beacon's frame of reference. If a beacon is unable to provide a bearing measurement, then the state of the robot is not fully observable and the state estimate will diverge. This is a common challenge in applications without access to persistent external measurement sources, such as underwater navigation, where extremely accurate proprioceptive sensors may be selected to slow the rate of state divergence over the course of a mission [13].

1.1.3 Motion Planning

Common motion planning approaches are presented by the authors of [14], [15] and [16], with the latter including a review of newer machine learning approaches. Grid-based approaches typically discretize the state space and employ a search to find a valid path to the goal. A* search is a popular extension of Dijkstra's algorithm that uses heuristics to guide search efforts from the origin toward the direction of the goal instead of exploring the entire environment. A major downside of the standard A* search algorithm is that any deviation from the path or a discovery of an unknown obstacle requires the entire path to be planned again, although there are strategies to recall previously computed path costs and adjust the single-source path as shown by the authors in [17]. Another approach is generating an artificial potential function that draws the system toward the goal state, with the authors of [18] demonstrating a hybrid technique of A* search and artificial potential to prevent the system from getting stuck in local minima.

However, due to the desire to apply model predictive control (MPC) to the planning problem, a solution to explore the state space was required. A hybrid approach that uses A* search to find an initial path which is then smoothed by MPC is presented by the authors of [19], but a wider variety of candidate paths was desired. Therefore, an implicit breadth-first is explored using the robot trajectory controller to develop a sequence of control inputs required to implement the state space search. This method also reduces the need for complex trajectories experienced by the authors of [20] when implementing motion planning for a robot with nonholonomic constraints such as our unicycle robot.

1.1.4 Model-Predictive Control

Model-predictive control is a type of predictive control that uses a known system model and cost function to optimize a sequence of control inputs over prediction time horizon T. The optimization becomes more challenging when applied to a nonlinear system, but there has been widespread research into the area of nonlinear model-predictive control (NMPC) due to the broad range of nonlinear control problems. Examples of model-predictive control applications range from health care and finance to power electronics and grid scale power systems as presented in [21]. Optimization functions may be solved analytically with tools such as nonlinear constrained optimization or solved using evolutionary algorithms that find global solutions to nonconvex problems [22]. In this research, a set of particle filters managed by a model-predictive controller will perform a search to capture the evolution of state dependent certainty based on observability.

1.2 Related Work

In motion planning, many approaches do not consider the state uncertainty that the system will face later on when the controller struggles to achieve the selected path. Therefore, several linear and nonlinear approaches to motion planning have been developed to also optimize state certainty with good results. Assumptions of Gaussian state uncertainty models are common due to the ease of applying optimization algorithms. For example, the authors of [23], [24] and [25] present similar approaches using rapidly-exploring random trees that simulate measurements in environments with restricted observability to track Gaussian state error along a path and find a trajectory that simultaneously avoids collisions and maximizes the probability of achieving the goal. The work in [26] expands the optimization factors to also minimize vehicle acceleration and maximize self calibration between inertial measurement unit (IMU) and visual sensors. The authors of [27] instead focus on achieving the goal while maximizing the norm of system Fisher information relating observable to unobservable characteristics, resulting in a system that generates control perturbations to observe system dynamics. In the case of higher degrees of freedom, the authors of [28] develop an observability aware system for the case of simultaneous localization and mapping (SLAM) but discount sampling-based planning approaches due to the computational demands.

Stepping away from analytical optimization functions, the distinction between motion planning and control becomes blurry with model-predictive control due to the inherent time horizon used for prediction. Specifically applied to motion planning, authors in [29] successfully applied model-predictive control to optimize factors like passenger comfort, safety, and speed of execution in autonomous vehicle maneuvering. Model-predictive control using particle filters has also been successfully demonstrated where a particle filter is evolved using the state transition model to determine the best set of control inputs. The approach taken by authors of [30] initialize MPC with single particle values, while other approaches taken by authors of [31] and [32] use the entire particle cloud for better state representation.

1.3 Summary of Contribution

This research demonstrates that a mobile ground robot operating in a two dimensional environment with pre-existing knowledge of sparse observable states can successful navigate to a goal while minimizing state estimate uncertainty. To accomplish this, a particle filter is used to estimate the state of the robot and similar particle filters are used to explore potential trajectories with simulated measurements to evaluate potential state uncertainty along each trajectory. Finally, a model-predictive control approach is used to score each potential path and select the most promising option, essentially performing the role of a higher level particle filter searching for the overall optimal path to the goal.

List of References

[1] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings of (ICRA) International Conference on Robotics and Automation*, vol. 2, May 1999, pp. 1322 – 1328.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[3] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 03 1960. [Online]. Available: https://doi.org/10.1115/1.3662552

[4] G. Kitagawa, "Non-gaussian state-space modeling of nonstationary time series," *Journal of the American Statistical Association*, vol. 82, no. 400, pp. 1032–1041, 1987. [Online]. Available: http://www.jstor.org/stable/2289375

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

[6] A. Doucet and A. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of Nonlinear Filtering*, vol. 12, 01 2009.

[7] C. Kuptametee and N. Aunsri, "A review of resampling techniques in particle filtering framework," *Measurement*, vol. 193, p. 110836, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263224122001312

[8] C. Snyder, "Particle filters, the optimal proposal and high-dimensional systems," Ph.D. dissertation, Shinfield Park, Reading, 2012 2012.

[9] R. Kalman, "On the general theory of control systems," *IFAC Proceedings Volumes*, vol. 1, no. 1, pp. 491–502, 1960, 1st International IFAC Congress on Automatic and Remote Control, Moscow, USSR, 1960. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017700948

[10] S. R. Kou, D. L. Elliott, and T. J. Tarn, "Observability of nonlinear systems," *Information and Control*, vol. 22, no. 1, pp. 89–99, 1973. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019995873905081

[11] R. Hermann and A. Krener, "Nonlinear controllability and observability," *IEEE Transactions on Automatic Control*, vol. 22, no. 5, pp. 728–740, 1977.

[12] A. Martinelli and R. Siegwart, "Observability analysis for mobile robot localization," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 1471–1476.

[13] J. Kinsey, R. Eustice, and L. Whitcomb, "A survey of underwater vehicle navigation: Recent advances and new challenges," 01 2006.

[14] D. Gonzalez Bautista, J. Pérez, V. Milanes, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, pp. 1–11, 11 2015.

[15] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," 01 2005.

[16] C. Zhou, B. Huang, and P. Fränti, "A review of motion planning algorithms for intelligent robotics," 2021.

[17] M. Barbehenn and S. Hutchinson, "Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 2, pp. 198–214, April 1995.

[18] H. Sang, Y. You, X. Sun, Y. Zhou, and F. Liu, "The hybrid path planning algorithm based on improved a* and artificial potential field for unmanned surface vehicle formations," *Ocean Engineering*, vol. 223, p. 108709, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002980182100144X

[19] K. Bergman, O. Ljungqvist, T. Glad, and D. Axehill, "An optimization-based receding horizon trajectory planning algorithm," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 550–15 557, 2020, 21st IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320330810

[20] A. Kelly and B. Nagy, “Reactive nonholonomic trajectory generation via parametric optimal control,” *The International Journal of Robotics Research*, vol. 22, no. 7-8, pp. 583–601, 2003. [Online]. Available: https://doi.org/10.1177/02783649030227008

[21] S. V. Raković and W. S. Levine, Eds., *Handbook of Model Predictive Control.* Birkhäuser Cham, 2019.

[22] I. Askari, S. Zeng, and H. Fang, “Nonlinear model predictive control based on constraint-aware particle filtering/smoothing,” in *2021 American Control Conference (ACC)*, May 2021, pp. 3532–3537.

[23] S. Prentice and N. Roy, “The belief roadmap: Efficient planning in linear pomdps by factoring the covariance,” in *International Symposium of Robotics Research*, 2007.

[24] A. Bry and N. Roy, “Rapidly-exploring random belief trees for motion planning under uncertainty,” in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 723–730.

[25] J. van den Berg, P. Abbeel, and K. Goldberg, “Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 895–913, 2011. [Online]. Available: https://doi.org/10.1177/0278364911406562

[26] C. Grebe, E. Wise, and J. Kelly, “Observability-aware trajectory optimization: Theory, viability, and state of the art,” in *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, Sep. 2021, pp. 1–8.

[27] A. D. Wilson, J. A. Schultz, and T. D. Murphey, “Trajectory synthesis for fisher information maximization,” *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1358–1370, Dec 2014.

[28] K. M. Frey, T. J. Steiner, and J. P. How, “Towards online observability-aware trajectory optimization for landmark-based estimators,” 2020.

[29] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng, “Path planning for autonomous vehicles using model predictive control,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 174–179.

[30] S. K. Botchu and S. Ungarala, “Nonlinear model predictive control based on sequential monte carlo state estimation,” *IFAC Proceedings Volumes*, vol. 40, no. 5, pp. 29–34, 2007, 8th IFAC Symposium on Dynamics and Control of Process Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015317663

[31] D. Stahl and J. Hauth, “Pf-mpc: Particle filter-model predictive control,” *Systems and Control Letters*, vol. 60, no. 8, pp. 632–643, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167691111001125

[32] J. de Villiers, S. Godsill, and S. Singh, “Particle predictive control,” *Journal of Statistical Planning and Inference*, vol. 141, no. 5, pp. 1753–1763, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378375810005252

CHAPTER 2

Methodology

2.1 Problem Setting

We assume that a mobile ground robot moves in a two-dimensional plane with a state described as position $[x, y]^T \in \mathbb{R}^2$ and orientation $\theta \in SO(2)$. The robot state evolves according to inputs linear velocity v and angular velocity ω following the nonlinear model for a unicycle:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{11}$$

We assume that a known number N of landmarks are present in the environment with the n^{th} landmark located at a known position $[x_l^{(n)}, y_l^{(n)}]^T \in \mathbb{R}^2$ and a maximum detection range of $R^{(n)}$. The robot obtains measurements of its position $[z_x^{(n)}, z_y^{(n)}]^T$ from the n^{th} landmark if the robot's range from the landmark $r^{(n)} < R^{(n)}$. If the robot is within detection range, the landmark will measure the robot's range $r^{(n)}$ and bearing $\alpha^{(n)}$ from the landmark. Based on the measurements, the landmark will determine the position of the robot according to the measurement equation:

$$\begin{bmatrix} x^{(n)} \\ y^{(n)} \end{bmatrix} = \begin{bmatrix} x_l^{(n)} + r^{(n)} * \cos\alpha^{(n)} \\ y_l^{(n)} + r^{(n)} * \sin\alpha^{(n)} \end{bmatrix} \tag{12}$$

The robot's task is to travel from a starting position $[x_s, y_s]^T$ to a known goal location $[x_g, y_g]^T$. Note that as long as the robot velocity $v > 0$, according to the discussion in section 1.1.2, the system is observable when the robot is within the detection range of at least one landmark, and is not observable otherwise.

2.2 System Architecture

A block scheme of the navigation system developed in this work is presented in Figure 2.1 and described below.

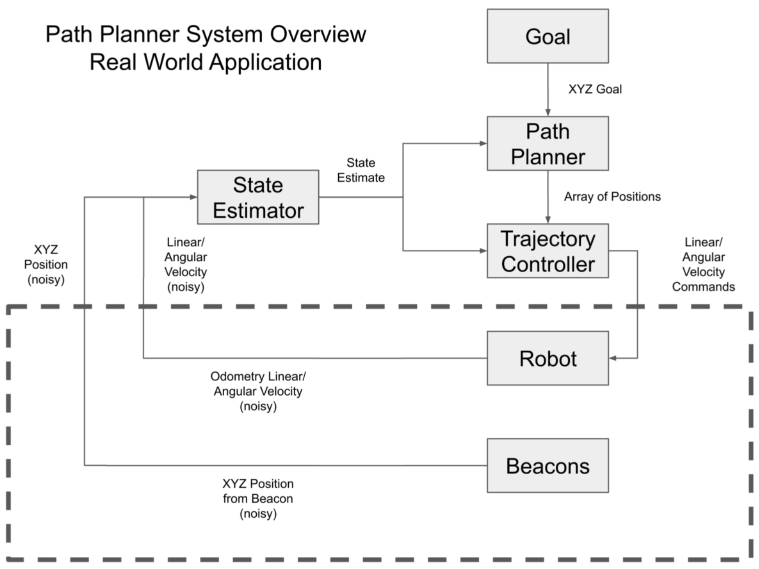
Path Planner System Overview
Real World Application
Goal
XYZ Goal
Path Planner
State Estimate
State Estimator
Array of Positions
Trajectory Controller
XYZ Position (noisy)
Linear/ Angular Velocity (noisy)
Linear/ Angular Velocity Commands
Robot
Odometry Linear/ Angular Velocity (noisy)
Beacons
XYZ Position from Beacon (noisy)

Figure 2.1: System Configuration Overview - Real World Application

2.2.1 State Estimator

Given the nonlinearity of the estimation problem, a standard implementation of the particle filter was selected for the state estimator. P particles are initialized with the state of each particle i identical to the robot state:

$$\begin{bmatrix} x^{(i)} \\ y^{(i)} \\ \theta^{(i)} \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{13}$$

and importance weights $w^{(i)} = \frac{1}{P}$ such that the sum of all weights is one.

$$\sum_{i=1}^{P} w^{(i)} = 1 \tag{14}$$

Given proprioceptive odometry measurements $[z_v, z_\omega]^T$, the state $[x^{(i)}, y^{(i)}, \theta^{(i)}]^T$ for each particle i is updated using the unicycle model from Equation 11 plus a noise sample $[V, \Omega]^T$ drawn from the actuation noise joint probability distribution function $\epsilon_{V,\Omega}(v, \omega)$.

$$\begin{bmatrix} \dot{x^{(i)}} \\ \dot{y^{(i)}} \\ \dot{\theta^{(i)}} \end{bmatrix} = \begin{bmatrix} \cos\theta^{(i)} & 0 \\ \sin\theta^{(i)} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_v + V \\ z_\omega + \Omega \end{bmatrix} \tag{15}$$

Note that the actuation applied to each particle is unique and therefore each particle will represent a hypothesis of the true state of the robot, with accuracy improving as the number of particles $P \to \infty$. The new state estimate of the robot is selected as the weighted sum of all particles states, with special care taken to find the weighted orientation average due to periodicity of θ on the interval $[0, 2\pi)$.

$$\begin{bmatrix} \overline{x} \\ \overline{y} \\ \overline{\theta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{P} w^{(i)} x^{(i)} \\ \sum_{i=1}^{P} w^{(i)} y^{(i)} \\ \text{atan2}\left(\sum_{i=1}^{P} w^{(i)} \sin\theta^{(i)}, \sum_{i=1}^{P} w^{(i)} \cos\theta^{(i)}\right) \end{bmatrix} \tag{16}$$

At an arbitrary time, an exteroceptive measurement $[z_x^{(n)}, z_y^{(n)}]^T$ may be received from an arbitrary beacon n. This measurement represents the sum of an ideally measured value and noise sampled randomly from a zero mean joint probability distribution $\delta_{X,Y}(x, y)$ characterizing the additive measurement noise. To calculate the likelihood that a particular particle hypothesis represents the true system state, an updated particle weight $w^{(i)'}$ is found by multiplying the previous particle weight by the value of the joint probability distribution function calculated for that particle's position error compared to the measured position.

$$w^{(i)'} = w^{(i)} * \delta_{X,Y}\left(x^{(i)} - z_x^{(n)}, y^{(i)} - z_y^{(n)}\right) \tag{17}$$

Finally, the weights of all particles are renormalized to sum to one, and this weight is used for each particle until the next measurement update re-weights each particle again.

$$w^{(i)''} = \frac{w^{(i)'}}{\sum_{i=1}^{P} w^{(i)'}} \tag{18}$$

After several re-weighting iterations, the issue of particle degeneracy may be encountered where the weights of outlying particles $w^{(i)} \rightarrow 0$ due to their poor representation of the state estimate. In this case, fewer particles will remain in the vicinity of the true state, so fewer particles will be available to represent the current state estimate, and therefore those particles will have higher weights to ensure the sum of all weights $\sum_{i=1}^{P} = 1$. This large variance in weights can be detected with an Effective Sample Size (ESS) metric computed using the sum of all particle weights squared.

$$ESS = \frac{1}{\sum_{i=1}^{P} {w^{(i)}}^2} \tag{19}$$

When each particle weight $w^{(i)} = \frac{1}{P}$, $ESS = P$ indicating no degeneracy as expected. However, as the particle weight variance increases, the value of ESS will drop and a threshold can be used to trigger an unbiased importance sampling of the particles to represent the same state estimate probability distribution but with equalized importance weights. One way to accomplish this is with systematic re-sampling where P linearly spaced values $r^{(k)}$ are generated

$$r^{(k)} = \frac{k}{P} + \frac{1}{2P}, k \in \{0, ..., P-1\} \tag{20}$$

and compared to the cumulative summation $cw^{(k)}$ of all particle weights.

$$cw^{(k)} = \sum_{i=1}^{k} w^{(i)}, k \in \{1, ..., P\} \tag{21}$$

Recall that the sum of all particle weights $\sum_{i=1}^{P} = 1$, so every randomly generated re-sampling point will correspond to specific particle, which is selected to construct a new posterior probability distribution. Particles that represent better state estimates will have higher weights and therefore occupy a larger portion of the cumulative summation, resulting in a higher likelihood of being re-sampled, approximately maintaining the posterior distribution after the re-sampling process.

One unique measurement opportunity afforded by an observability map approach to path planning is the intrinsic boolean information of whether the robot is inside or outside of an observable region. Therefore, a similar approach to measurement updates and re-sampling is applied to this additional measurement source. If a set of particles $\mathcal{P} \subset \{i = 1, ..., P\}$ is found to be in conflict with the current measurement availability (e.g. a measurement was not received, but one or more particles existed within an observable region), then those conflicting particles are resampled to remove their contribution from the state estimate. This was accomplished by setting the weights $w^{(i)}$ of the conflicting particles to zero and re-sampling.

$$w^{(i)} = 0, \forall i \in \mathcal{P} \tag{22}$$

This approach is similar to terrain aided navigation as discussed by the authors of [1] which can improve robot state estimation performance. For example, if the robot intended to enter an observable region, but narrowly missed due to state estimate error, the particles in the observable region representing an incorrect hypothesis are pruned from the particle filter, causing the weighted state estimate of the new particle distribution to converge toward the ground truth. Due to issues with sporadic performance near observability thresholds, particle pruning was limited to 30% which yielded suitable performance. The impact of particle pruning can be seen to improve the robot state estimate in the sample shown in Figure 2.2.

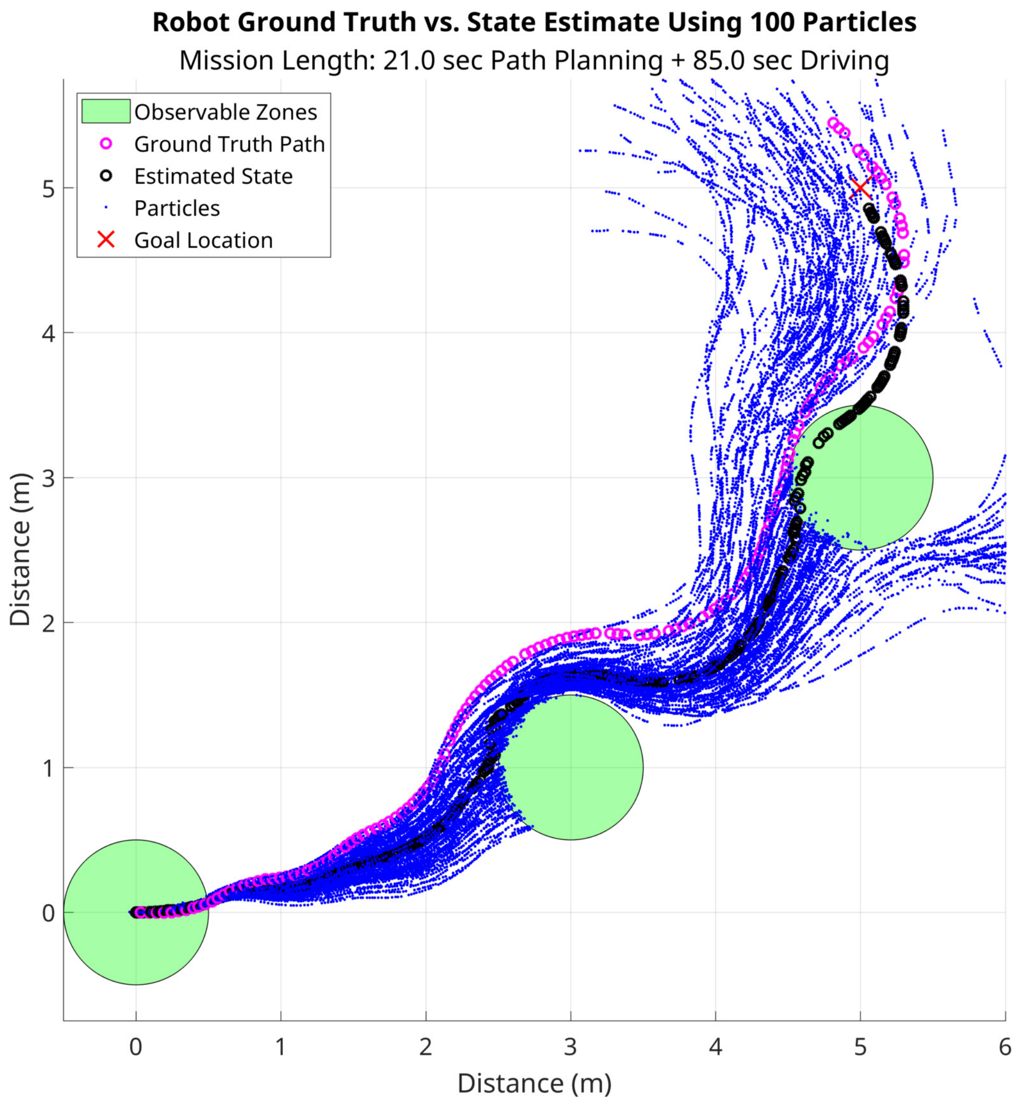

Figure 2.2: Particle Pruning Using "Negative Measurement" Knowledge

Similarly, when the robot is inside an observable area receiving measurements, particles existing outside the observable region are pruned, but this was rarely needed in practice because the state estimate quickly converged when the robot was collecting measurements in an observable region.

2.2.2 Trajectory Controller

A differential drive robot platform is one of the simplest options for a mobile ground robot because it only has two actuators - the left and right wheels. This type of robot is capable of moving forwards, moving backwards, turning in place, or turning while in motion given any combination of positive or negative control inputs for linear velocity v and angular velocity ω. Due to these motion paths, this type of configuration can be represented by a unicycle model with the kinematic model previously expressed in Equation 11.

Note that the system is nonholonomic in that it does not allow sideways motion, so to follow an arbitrary trajectory a control point B must be considered a distance b in front of the midpoint of the wheels. With this control point we find an input-output feedback linearization controller using the procedure found in [2]. First, the kinematic model is rewritten in terms of the new control point B:

$$\begin{bmatrix} \dot{x}_B \\ \dot{y}_B \end{bmatrix} = \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{23}$$

and a transformation matrix is defined as

$$T(\theta) = \begin{bmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \end{bmatrix}, \text{such that} \begin{bmatrix} \dot{x}_B \\ \dot{y}_B \end{bmatrix} = T(\theta) \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{24}$$

To guarantee the existence of $T^{-1}(\theta)$, $T(\theta)$ must not be singular, and in this case

$$det(T(\theta)) = \begin{vmatrix} \cos\theta & -b\sin\theta \\ \sin\theta & b\cos\theta \end{vmatrix} = b\cos^2\theta + b\sin^2\theta = b \tag{25}$$

As long as $b \neq 0$, $T^{-1}(\theta)$ exists which highlights the need for a control point B that is not directly between the robot wheels. Equation 24 is therefore inverted to solve for the control input:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = T^{-1}(\theta) \begin{bmatrix} \dot{x}_B \\ \dot{y}_B \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\frac{\sin\theta}{b} & \frac{\cos\theta}{b} \end{bmatrix} \begin{bmatrix} \dot{x}_B \\ \dot{y}_B \end{bmatrix} \tag{26}$$

and the system in equation 11 is re-written to form:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\frac{\sin\theta}{b} & \frac{\cos\theta}{b} \end{bmatrix} \begin{bmatrix} \dot{x}_B \\ \dot{y}_B \end{bmatrix} \tag{27}$$

Now finding $\dot{\theta}$ we see that

$$\dot{\theta} = -\frac{\dot{x}_B \sin\theta}{b} + \frac{\dot{y}_B \cos\theta}{b} = \frac{\dot{y}_B \cos\theta - \dot{x}_B \sin\theta}{b} \tag{28}$$

where we can define $\dot{x}_B$ and $\dot{y}_B$ as inputs to the new linear system

$$\begin{aligned} \dot{x}_B &= u_1 \\ \dot{y}_B &= u_2 \\ \dot{\theta} &= \frac{u_2 \cos\theta - u_1 \sin\theta}{b} \end{aligned} \tag{29}$$

where x_B and y_B act as integrators. Now we can select a desired trajectory composed of a desired position $q_d(t)$ and desired velocity $\dot{q}_d(t)$ for all time in our planned trajectory $t \in [0, T_d]$ where:

$$\begin{aligned} q_d(t) &= \begin{bmatrix} x_d(t) \\ y_d(t) \end{bmatrix}, t \in [0, T_d] \\ \dot{q}_d(t) &= \begin{bmatrix} \dot{x}_d(t) \\ \dot{y}_d(t) \end{bmatrix}, t \in [0, T_d] \end{aligned} \tag{30}$$

Now a control law for system 29 can be implemented as:

$$\begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} \dot{x}_d(t) \\ \dot{y}_d(t) \end{bmatrix} + \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \left(\begin{bmatrix} x_d(t) \\ y_d(t) \end{bmatrix} - \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \right) \tag{31}$$

with arbitrary positive controller gains k_x and k_y. Finally, using the relationship established in the system 29, the computed values u_1 and u_2 are applied to the inverted system equation 26 to calculate the actual input for the robot:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\frac{\sin\theta}{b} & \frac{\cos\theta}{b} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{32}$$

Note that with this control method, the desired trajectory can be arbitrary at the cost of losing direct control over the robot orientation θ.

To minimize the scope of each class needed for functionality, the Unicycle robot class implements the input-output linearization feedback controller for the unicycle model derived above. When running the system in "naive navigation" mode, the controller drives the state of the robot directly to the goal, otherwise the path planner is used for higher level trajectory planning and robot control only operates locally to drive the robot to the next point along the path.

Each path is constructed from a time stamped seed location and subsequent positions incremented by a fixed time step. The robot selects the next timestamped position into the future as the desired position $q_d[k]$ to pursue. Additionally, feed-forward velocity $\dot{q}_d[k]$ is calculated from the difference in adjacent positions along path divided by the path time step as shown below.

$$\dot{q}_d[k] = \frac{q_d[k] - q_d[k-1]}{q_t[k] - q_t[k-1]} \tag{33}$$

The trajectory controller then uses the desired trajectory to drive the robot toward that position, continuously iterating through the sequence of path positions until a new path is received. Care has been taken in this research to make the transition between paths as seamless as possible, but the transition could be a large discontinuity, especially with long path planning cycles.

Once the robot is within a certain distance from the goal, the robot switches from the path trajectory to simply using the goal as the next target. In this research a distance of 0.5 m was used as the threshold to shift the controller from "path following" to "goal seeking" by using the values below.

$$\begin{aligned} q_d[k] &= \begin{bmatrix} x_g \\ y_g \end{bmatrix} (m) \\ \dot{q}_d[k] &= \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix} (\tfrac{m}{s}) \end{aligned} \tag{34}$$

2.3 Model-Predictive Control

In this work, model-predictive control was implemented to selected a "best path" by weighting a time varying blend of scoring factors. Each potential path was evaluated by propagating a particle filter with a reduced particle count, but sampled from a multivariate Gaussian to roughly preserve current state estimate uncertainty characteristics. Furthermore, the implementation of the path planning method was designed to mimic the operation of a particle filter operating at a higher level:

- Propagating the paths outward increases the variance of path endpoints at each path planning step k, representing a sample of all possible paths the robot could take.
- A scoring function performs a measurement of the relative performance of each path, updating the weight of each path for each time step k.
- Once the path planner reaches the time horizon, the best path is selected using the highest weight which causes that path to be resampled as the initial state for all paths in the next iteration of path planning.

Two factors were used to achieve satisfactory path weighting performance in this research - normalized distance to the goal and a state estimate covariance metric. Given the goal location (x_g, y_g), the normalized distance to the goal nd was computed by taking the euclidean distance d of each path endpoint $x_k^{(i)}, y_k^{(i)}$ to the goal and normalizing the distance with respect to the mean of all M path endpoint distances. This factor was designed to give values for nd that were closely clustered around 1.0 when the paths were far from the goal, allowing the paths to explore without significant penalty. Once the paths drew closer to the goal, the path endpoints would become more distinguished and allow for easy selection of high performers.

$$d_k^{(i)} = \sqrt{\left(x_g - x_k^{(i)}\right)^2 + \left(y_g - y_k^{(i)}\right)^2} \tag{35}$$

$$nd_k^{(i)} = \frac{d_t^{(i)}}{\frac{\sum_{i=1}^{M} d_k^{(i)}}{M}} \tag{36}$$

Secondly, given the covariance matrix $\Sigma_k^{(i)}$ of the particle positions for the particle filter exploring path i, a metric c was developed to give an absolute measure of state estimate uncertainty. Conceptually, the diagonal eigenvalue matrix $\Lambda_k^{(i)}$ is computed and the eigenvalues along the diagonal are squared and summed to yield a single value for particle filter i. In practice, a more computationally efficient way of finding this value is taking the trace of the square of the covariance matrix.

$$c_k^{(i)} = tr\left({\Sigma_k^{(i)}}^2\right) \tag{37}$$

The progress of time toward the time horizon for each path planning iteration was normalized on the interval $0 <= t <= 1$ to serve as a time variant scaling factor independent of the user configurable time horizon. The overall scoring function uses a negative exponential function to generate a large value for over performing paths, and a small value with less variation for under performing paths. On top of choosing the exponential decay term to weight each factor, the time value weights the covariance factor at a steadily increasing amount, while the distance factor is heavily weighted at the end of the time horizon by raising it to the 6^{th} power.

These weighting functions were selected to accumulate path performance over the entire time horizon as opposed to a strategy that only looks at the endpoint. If only the endpoint was considered, a potential path may appear to be a strong candidate, but if the projected path encountered an observable zone near the end of the time horizon there is no guarantee that the robot could actually reach that observable zone due to state estimate divergence. Instead, a robot that travels through an observable zone near the middle of the time horizon may yield better results because there is a higher certainty of the robot reaching those observable states, even if the state estimate diverges again slightly before the time horizon. The complete function that computes each path score $s_k^{(i)}$ is shown below and the weight factors functions are shown in Figure 2.3.

$$s_k^{(i)} = 2t^6 \exp\left(-nd_k^{(i)}\right) + 1.5t \exp\left(-2c_k^{(i)}\right) \tag{38}$$

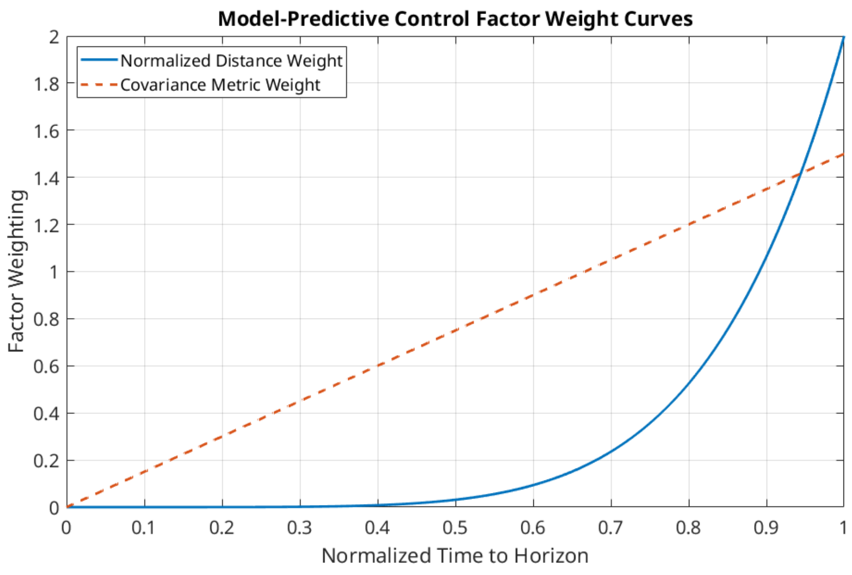

Figure 2.3: Model-Predictive Control Factor Weighting Curves

Finally, the score of each path is normalized for that time interval and added to running sum of the performance weight for each path $w_k^{(i)}$.

$$w_k^{(i)} = w_{k-1}^{(i)} + \frac{s_k^{(i)}}{\sum_{i=1}^{M} s_k^{(i)}} \tag{39}$$

Note that the full reward system was selected to be positive and bounded. However, the final product delivered by this method is a set of positions that form a path, as opposed to true model-predictive control which applies the optimized control inputs directly to the plant. This deviation was intentional due to the long computation times needed for each path planning iteration. In the case of this research, waiting for the path planner response to an updated state estimate would cause a response delay of 20 seconds or more.

Fortunately, the method used in this research does preserve the model evolution characteristics as illustrated previously in Figure 2.4. A target somewhat behind the robot will cause the low level controller to initially back up the robot and turn before driving forward. That behavior is seen in the top three paths, whereas the fourth and adjacent paths immediately drive forward, initially taking the same route due to control input saturation. These model specific control input realizations are preserved in the generated path and carried out by the path follower.

2.3.1 State Space Exploration

One major breakthrough that enabled the success of this research was efficiently generating a sequence of control inputs to explore a wide variety of potential paths. A breadth-first search of the state space was implemented by defining an "explore angle" γ_e for all M paths to fan out and explore. The sequence of control inputs to explore this variety of paths was generated by assigning each path i a linearly spaced angular offset $\alpha^{(i)}$ centered around 0:

$$\alpha^{(i)} = -\frac{\gamma_e}{2} + \frac{\gamma_e}{2M} + \frac{i}{M}, i \in \{0, ..., M-1\} \tag{40}$$

and placing a new fictitious goal point $[x_g^{(i)}, y_g^{(i)}]^T$ that was unique to each path given the path's assigned angular offset $\alpha^{(i)}$ from the goal direction. This was accomplished by applying a rotation matrix $R_g^{(i)}$ to the vector error between each path endpoint $[x^{(i)}, y^{(i)}]^T$ and the goal, then adding the resulting vector back to the path endpoint.

$$R_g^{(i)} = \begin{bmatrix} \cos\alpha^{(i)} & -\sin\alpha^{(i)} \\ \sin\alpha^{(i)} & \cos\alpha^{(i)} \end{bmatrix} \tag{41}$$

$$\begin{bmatrix} x_g^{(i)} \\ y_g^{(i)} \end{bmatrix} = R_g^{(i)} \left(\begin{bmatrix} x_g \\ y_g \end{bmatrix} - \begin{bmatrix} x^{(i)} \\ y^{(i)} \end{bmatrix} \right) + \begin{bmatrix} x^{(i)} \\ y^{(i)} \end{bmatrix} \tag{42}$$

Each individualized goal point is pursued by the low level controller, then over multiple time steps the resulting paths form a fan-shaped projection that ultimately seek out and wrap around the goal as seen in Figure 2.4.

2.3.2 Path Seed Projection

Two challenges of our resource constrained path planning were misalignment between adjacent paths and running out of new path positions to pursue due to the computation time needed to plan each path. To mostly overcome these limitations, all new paths were seeded in the future by projecting the current state forward in time. This was accomplished by spawning a new particle filter from the current pose estimate of the robot and sampling particles from a multivariate Gaussian distribution to simulate the current state estimate uncertainty. Using the previously published path and the time it took to plan the last path, the path planner simulated the control inputs and measurements for an ideal robot to project the new particle filter forward in time.

The resulting particle filter theoretically captured the expected state of the robot at the future time when the new path became ready, so it was used as a seed for the new iteration of path exploration. In practice, best performance

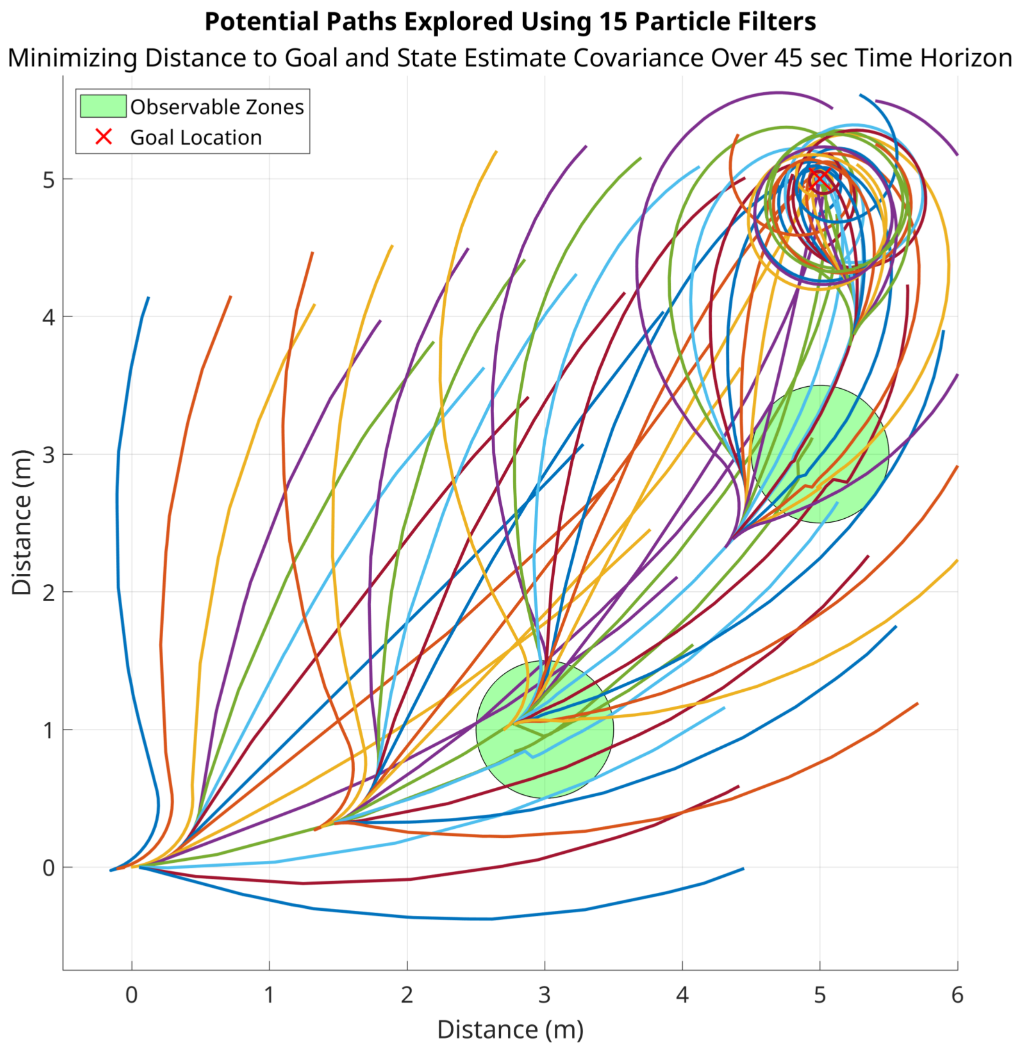

Figure 2.4: Potential Paths Explored Using a $\frac{3}{4}\pi$ Exploration Angle

was achieved when the path was projected to about 90% of the expected time to complete the path planning to leave a margin for inconsistency in computation time. Because the low level controller followed a path rather than directly applying a sequence of commands, small offsets between similarly oriented paths were well tolerated with the robot asymptotically approaching the new path due to feed-forward control.

List of References

[1] J. Melo and A. Matos, “Survey on advances on terrain based navigation for autonomous underwater vehicles,” *Ocean Engineering*, vol. 139, pp. 250–264, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002980181730241X

[2] H. Marquez, *Nonlinear Control Systems: Analysis and Design.* Wiley-Interscience, 11 2002.

CHAPTER 3

Implementation

3.1 Robot Platform

For simplicity, the Cherokey mobile robot platform was selected for this research and configured with differential drive to emulate the ”unicycle” model. The Cherokey platform, shown in Figure 3.1, had two motorized wheels controlled by a microcontroller board running the low level wheel speed controllers with encoder feedback delivered at 5 Hz. The vehicle was about 225 mm long and 175 mm wide with a maximum speed limited to $\pm 10.0\ \frac{m}{s}$ and maximum rotation of $\pm 0.2\ \frac{rad}{s}$ for these trials. Each wheel rotation gave about 20 cm of forward travel resulting in a wheel encoder resolution of 96 $\frac{ticks}{cm}$.

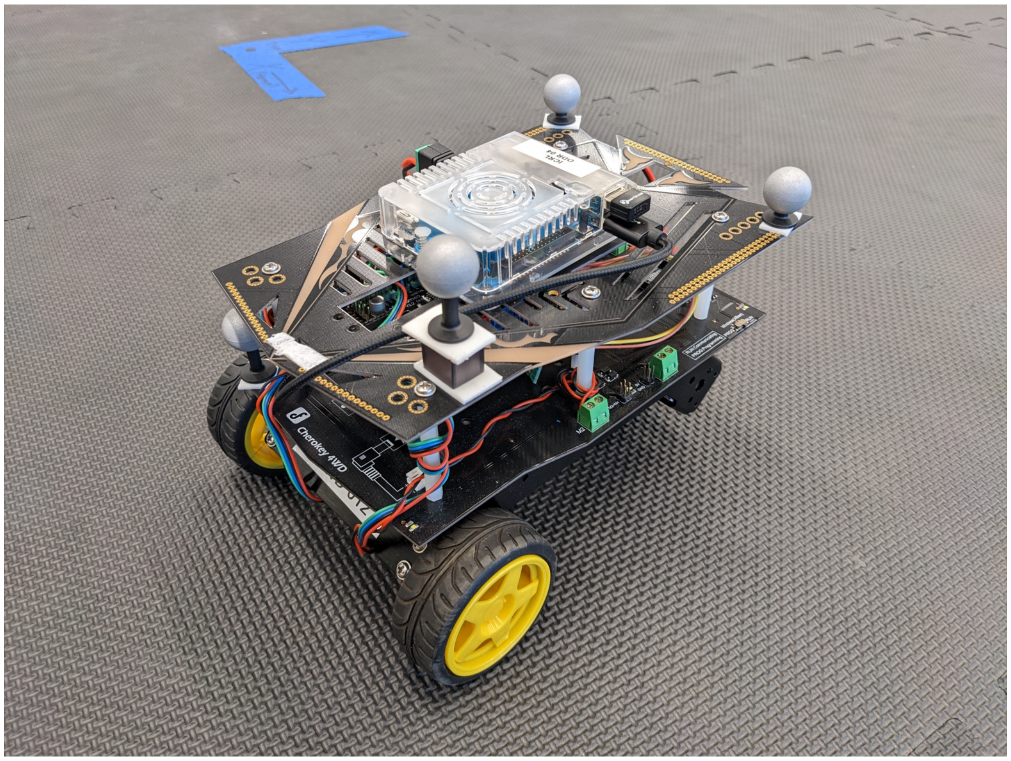
Figure 3.1: Cherokey Mobile Ground Robot

Noise characteristics of the vehicle were assumed to be Gaussian and estimated empirically by comparing odometry wheel encoder integration to ground truth position measurements. In this research, actuation noise is added to the linear velocity v and angular velocity ω with the following mean μ and variance σ:

$$\begin{aligned} \begin{bmatrix} \mu_v \\ \mu_\omega \end{bmatrix} &= \begin{bmatrix} 0.0\ \frac{m}{s} \\ 0.0\ \frac{rad}{s} \end{bmatrix} \\ \begin{bmatrix} \sigma_v^2 & \sigma_v * \sigma_\omega \\ \sigma_v * \sigma_\omega & \sigma_\omega^2 \end{bmatrix} &= \begin{bmatrix} 0.002 & 0.0 \\ 0.0 & 0.005 \end{bmatrix} \end{aligned} \tag{43}$$

3.2 OptiTrack Motion Capture System

For real time ground truth feedback, an OptiTrack Motion Capture system was configured to track the robot as a rigid body and stream real time positions at a rate of 2 Hz. For the purpose of these experiments, positions were assumed to be in the world frame as defined during the most recent OptiTrack calibration. A view of the OptiTrack screen during an experiment is shown in Figure 3.2 with the reflective markers grouped as the "robot" object.

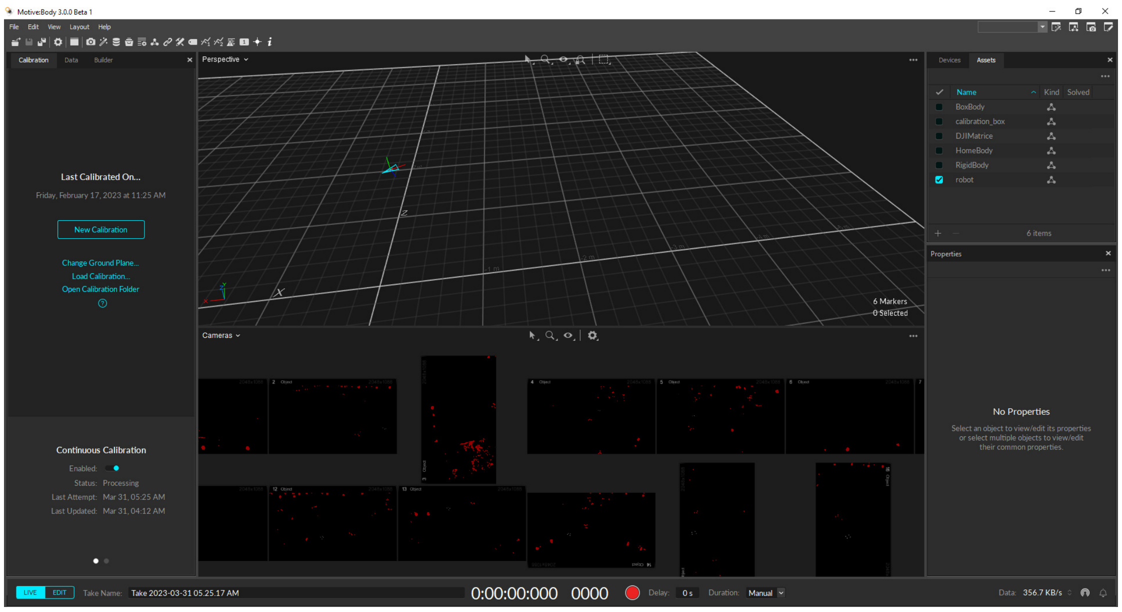

Figure 3.2: OptiTrack Motion Capture User Interface

To simulate the measurement quality of a real beacon, each measurement passed to the ground robot state estimator was sampled from a multinomial Gaussian distribution using the OptiTrack position for μ_m and covariance:

$$\Sigma_m = \begin{bmatrix} 0.001 & 0.0 \\ 0.0 & 0.001 \end{bmatrix} \tag{44}$$

3.3 Computation

An Odroid single board computer with Wi-Fi dongle provided the Cherokey robot's onboard network communication link to interact with local information streams. The Odroid communicated with the wheel driver microcontroller board over Universal Serial Bus (USB) to send linear and angular velocity commands, then receive the linear and angular velocity inferred from the wheel encoders for state estimation.

Most computation for this project was completed on a personal laptop computer with a 4th generation Intel i5-4210u mobile processor. For simulations, the laptop computer was able to support a 0.8 real time factor and give satisfactory results for prototyping and testing the system. For simulation validation with the real robot, the laptop computer gave slightly better results in real time, potentially due to the lack of simulation load. This computational performance is in alignment with the vision of this path planning method running in real time on a mobile ground robot.

The Python scripting language was used to implement all the software that performed the primary tasks of this thesis research. Additional scripts in Matlab/Octave and bash were used to generate the observability maps, automate simulation trials, and interpret results. Generally, the software was designed to be as universal as possible, with a Unicycle object class providing the main functionality for any Unicycle specific state and input variable modifications. By limiting the scope of functionality for each piece of software, the particle filter and path planner implementations could be generalized to three dimensions and pass variables without the need to understand data structures specific to the Unicycle object.

3.4 Robot Operating System Configuration

Robot Operating System (ROS) is a collection of software libraries designed to streamline the development and deployment of robotic systems. One unique aspect of ROS is the use of a network based communication with a publisher/subscriber model using ROS "topics" as a way for ROS nodes to communicate. This allows for the development of distinct nodes that perform the main functionality of the system, but also allows for nodes to simulate various aspects of the anticipated application that can be bypassed once deployed. In this case, the Gazebo simulator was used to simulate the physical process of driving a small mobile ground robot, similar to the real world application, and publish a ROS topic that would be contextually similar.

Additionally, passing predefined messages on ROS topics over the local network allowed the nodes to exist anywhere as long as they had access to the network and knew which machine was running the ROS core. In this work, the various python scripts were designed to have distinct functionality and communicate through ROS topics for easy integration with real world applications. Specifically, a laptop computer was used to run all the nodes for the simulation, and the simulator was seamlessly replaced with the ROS enabled Cherokey robot and OptiTrack Motion Capture System when experimental trials were conducted to validate simulation results. In future work it would be trivial to run the computational intensive path planner on a higher performance machine and publish the resulting paths on the appropriate ROS topic for the robot to follow.

To improve ease of use, ROS features a "roslaunch" file format that allows users to predefine a desired ROS environment and launch specific nodes simultaneously using an XML ".launch" file. In the case of this research, four launch files were created to cover a variety of test conditions. A pair of launch files were created to either run the software on the Gazebo simulator or in the real world with the ground robot and OptiTrack Motion Capture System. Then, each launch file was copied to either launch a trial of the path planner software or launch a trial of the naive navigation. In each launch file, specific nodes were included as needed to activate the desired functionality for a complete system.

3.4.1 Goal Publisher

To communicate the current goal position, a goal node was created for the sole purpose of publishing the desired goal whenever queried. This gave flexibility for the user to latch a new desired goal message to change robot tasking during operation. In the current implementation, any changes to the goal will be incorporated when the path planner started the next path planning iteration. The goal was published as a standard Pose message and used continuously by the path planner node and near the end of the trials by the control node to use simple control once within a short distance of the goal.

3.4.2 Noisy Odometry Simulator

For dead reckoning, the robot wheel encoders were used to estimate the vehicle's linear and angular velocity. However, for the purpose of simulation, the standard Odometry message published by the Gazebo simulator was noise free and did not represent a realistic case. Therefore, a noisy odometry simulation node was created to take the ideal odometry values and add actuation noise with the characteristics presented in Subsection 3.1 to make system simulation more realistic. To transition the system to the real world operating on a real robot, this node was simply bypassed so the mobile ground robot could directly provide noisy odometry input to the state estimator. Noisy odometry was published as a standard Odometry message and used by the state estimator node.

3.4.3 Measurement Simulator

A measurement source was required for the state estimator to reduce uncertainty during robot operation. In the real application, measurements could be derived in any number of ways from a set of beacons, but for this research the simple case of bearing and range from the beacon was selected to yield full observability as shown in [1]. Measurements from Gazebo or the Optitrack Motion Capture system were considered ideal ground truth measurements, so noise was added as described above in Subsection 3.2. Measurements were published as a standard Odometry message, to accommodate position and velocity components if needed, and used by the state estimator node.

3.4.4 State Estimator

The state estimator node implemented the localization particle filter described in Subsection 2.2.1. The particle filter class consisted of higher level functions that used the robot object to perform the time update and measurement update phases. Re-sampling was conducted autonomously at the particle filter level by using state manipulation functions provided by the robot class. The state estimation particle filter used in this research employed 100 particles and re-sampled the particles once the effective sample size computed using Equation 19 reached 50% of the original particle number, in this case 50 particles. In software, this was implemented as 100 copies of the Unicycle robot object with references to the original robot object broken to allow for independent motion and state management of each particle.

This node was subscribed to the noisy odometry sensor feedback and measurement source. A new state estimate was published each time a new noisy odometry message was received. The state estimate was published as a standard Odometry message to accommodate position and velocity components if needed, and used by the controller for navigation and the path planner to seed the next path.

3.4.5 Path Planner

As the most computationally intensive portion of the project, the path planner node subscribed to the state estimate messages, but was configured to keep an empty queue so the state estimate was up to date whenever the path planner node was ready to seed a new path. This node implemented the model-predictive controller and associated performance enhancements described in Section 2.3. The completed path was packaged as a standard PoseArray message and published for the trajectory controller to follow.

3.4.6 Trajectory Controller

The trajectory controller node implemented the path following described in Subsection 2.2.2 and applied conservative saturation values to all commands to prevent damage to the robot. This node was subscribed to the path planner to maintain the most recent path, the state estimator to determine an updated control command to reach the next planned path position, and the goal publisher to switch trajectory control once the robot was in the vicinity of the goal.

List of References

[1] A. Martinelli and R. Siegwart, "Observability analysis for mobile robot localization," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 1471–1476.

CHAPTER 4

System Simulation

To reduce the development time for this research, the Gazebo simulator was used in conjunction with Robot Operating System (ROS) to simulate the performance of the path planning system compared to a naive navigator that does not use knowledge of observability to aid in the collection of high quality position measurements. The configuration of the system using the Gazebo simulation is shown in Figure 4.1.

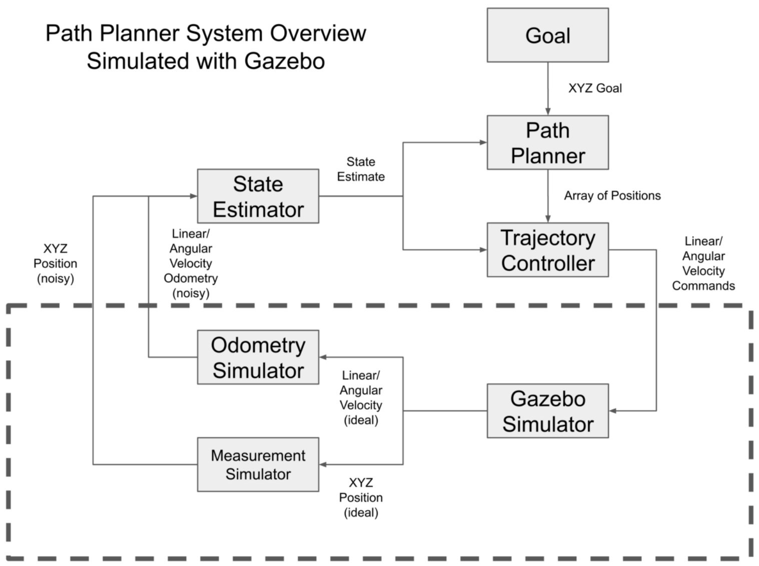

Figure 4.1: System Configuration Overview - Gazebo Simulation

4.1 Simulation Setup

Gazebo generated ideal odometry feedback and ideal position measurements, so the simulated robot in Gazebo was used as the ground truth and the output streams were modified with added noise quantified in Section 3.1 to more accurately simulate real world odometry and measurement quality. ROS nodes described in Section 3.4 were launched to perform the following simulations and generate results.

4.2 Simulation Results

4.2.1 Case of Full Observability

The trivial case for this research must compare the performance of the path planner against naive navigation in an environment with global full observability to ensure there is no loss in standard functionality. As expected, the end result of both approaches is indistinguishable as shown in Figure 4.2 and Figure 4.3. The main exception is the additional time required for the path planner to generate the initial path before embarking.

As expected, neither approach varied by much from the ground truth. A comparison showing the state estimate error for the path planner and naive navigation is shown in Figure 4.4 and Figure 4.5 respectively.

4.2.2 Case Study #1

The first case study demonstrates the preference for traveling in an observable zone due to the resulting higher state estimate confidence. In this study, the goal was located at (5,5) and a row of observable zones was positioned to the side of the direct path to the goal, each with a radius of 0.5 m and located at (2,0), (3,1), (4,2), and (5,3).

In the case of the naive controller heading directly for the the goal, the state estimate will steadily diverge unless the robot happens to come close enough or encounter an observable zone and the state estimate converges. An example of an accidental success of the naive navigation is shown in in Figure 4.6.

In sharp contrast, the path planner will intentionally seek out observable zones on the way to the goal in order to maintain a low state estimate uncertainty. The sequence of best paths selected during the path planning operation is shown in

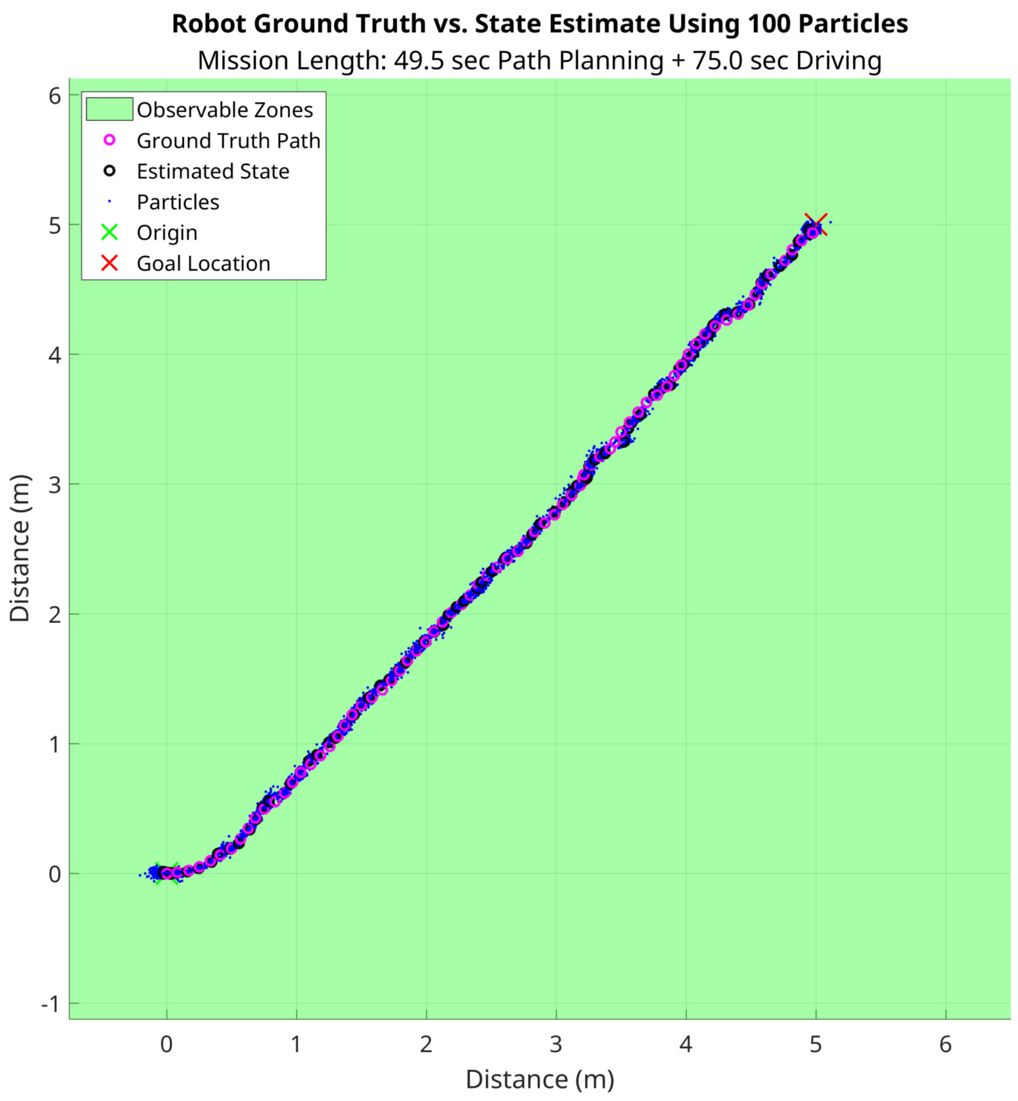

Figure 4.2: Global Full Observability Path Planner Performance

Figure 4.7, with the proposed path and state estimate for the path planner robot shown in Figure 4.8.

This comparison of simulation results was repeated 101 times to see the distribution of final error that could be expected in a real trial. Paths for both sets of simulations are overlaid in Figure 4.9 with the final robot positions shown in Figure 4.10. The covariances of final positions for each method of navigation were computed to draw 95% confidence ellipses indicating a clear improvement of

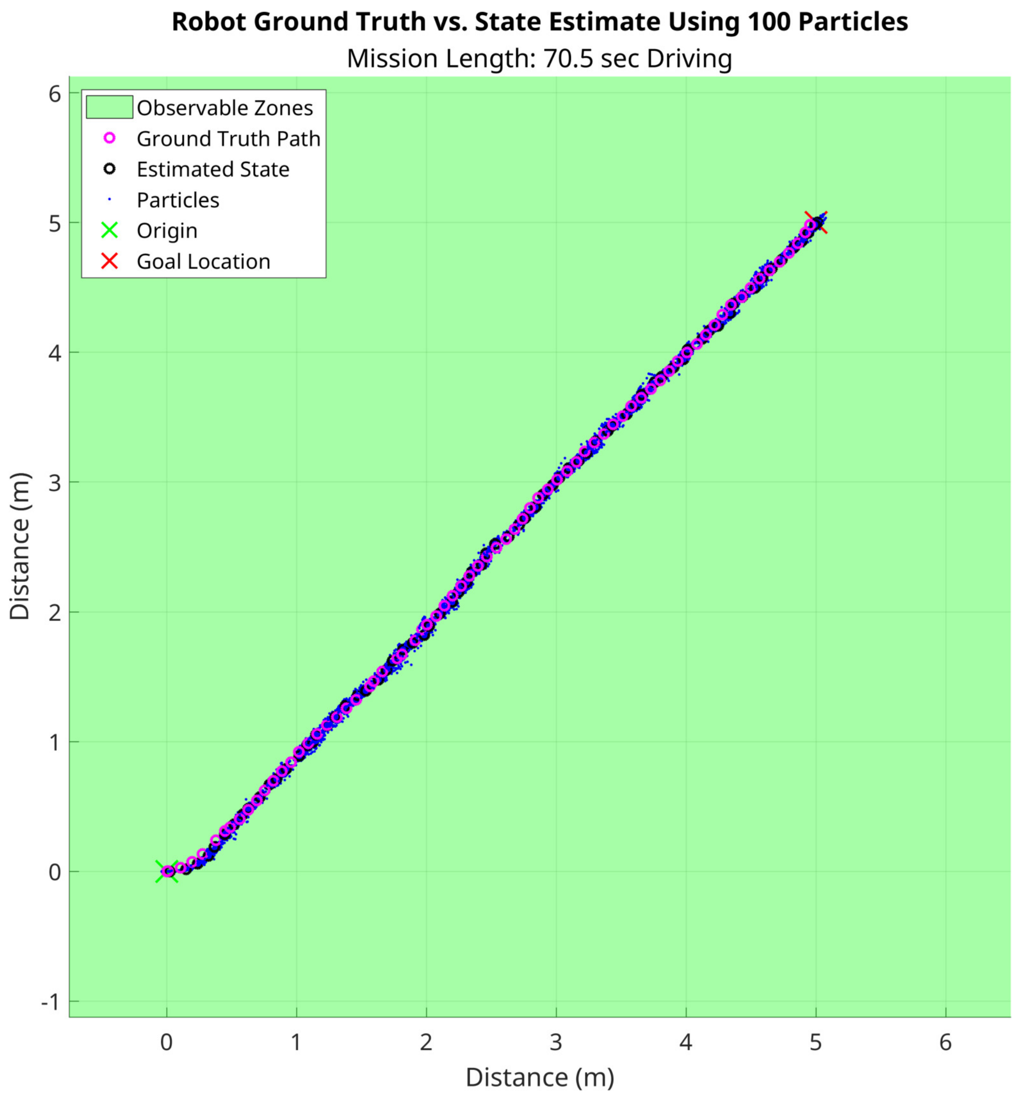

Figure 4.3: Global Full Observability Naive Navigation Performance

the path planner over naive navigation in this scenario. Table 1 shows the state estimate error and time requirements for each approach.

4.2.3 Case Study #2

The second case study demonstrates the limitations of the path planner when the environment was so sparsely populated with observable zones that the robot may not be able to navigate as desired. In this study, the goal was also located at (5,5) and two observable zones were positioned to one side of the direct path to

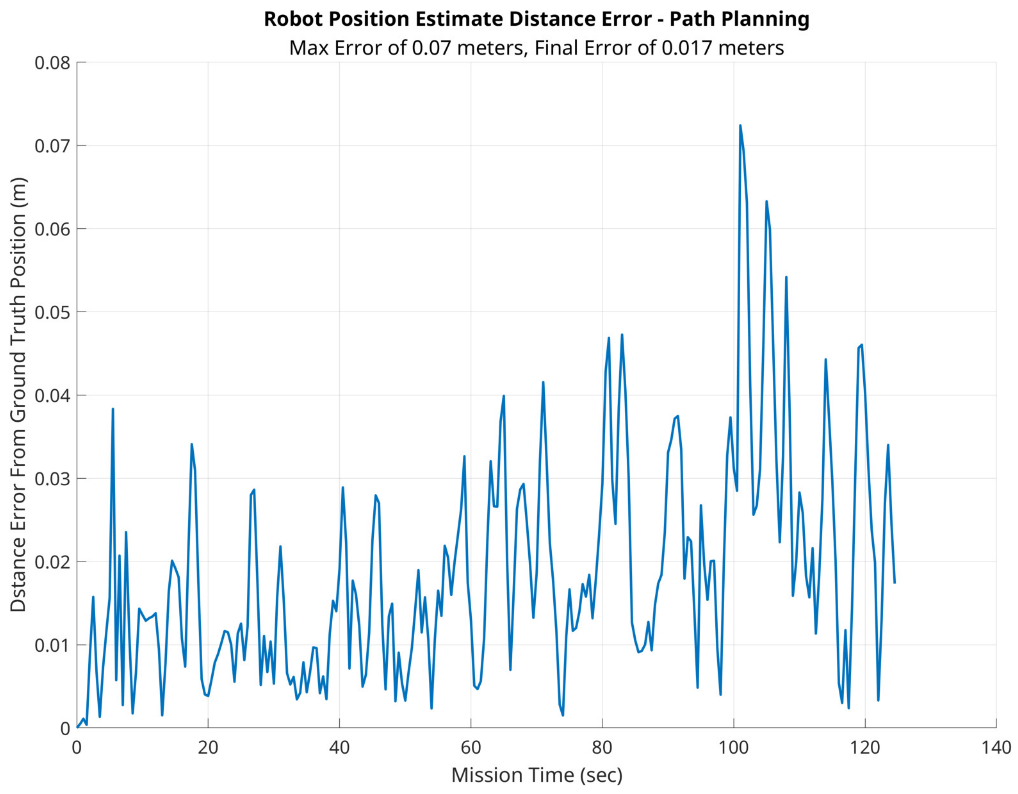

Figure 4.4: Path Planner State Estimate Error from Ground Truth

Statistic	Calculation Method	Naive Navigator	Path Planner
Position Estimate Error en Route (m)	Absolute Mean Value	0.282	0.047
	Root Mean Square	0.535	0.098
Final Position Error (m)	Absolute Mean Value	0.719	0.222
	Root Mean Square	0.904	0.270
Initialization Time (s)	Absolute Mean Value	1.693	17.15
	Root Mean Square	1.715	17.43
Drive Time (s)	Absolute Mean Value	73.39	81.59
	Root Mean Square	73.43	81.63

Table 1: Naive Navigation vs. Path Planner Statistics for Case #1 (101 Trials)

the goal, each with a radius of 0.5 m and located at (3,1) and (5,3).

In the case of the naive controller heading directly for the goal, the state estimate again steadily diverged unless the robot happened to encounter an observable zone. A typical example of the naive navigation is shown in Figure 4.11.

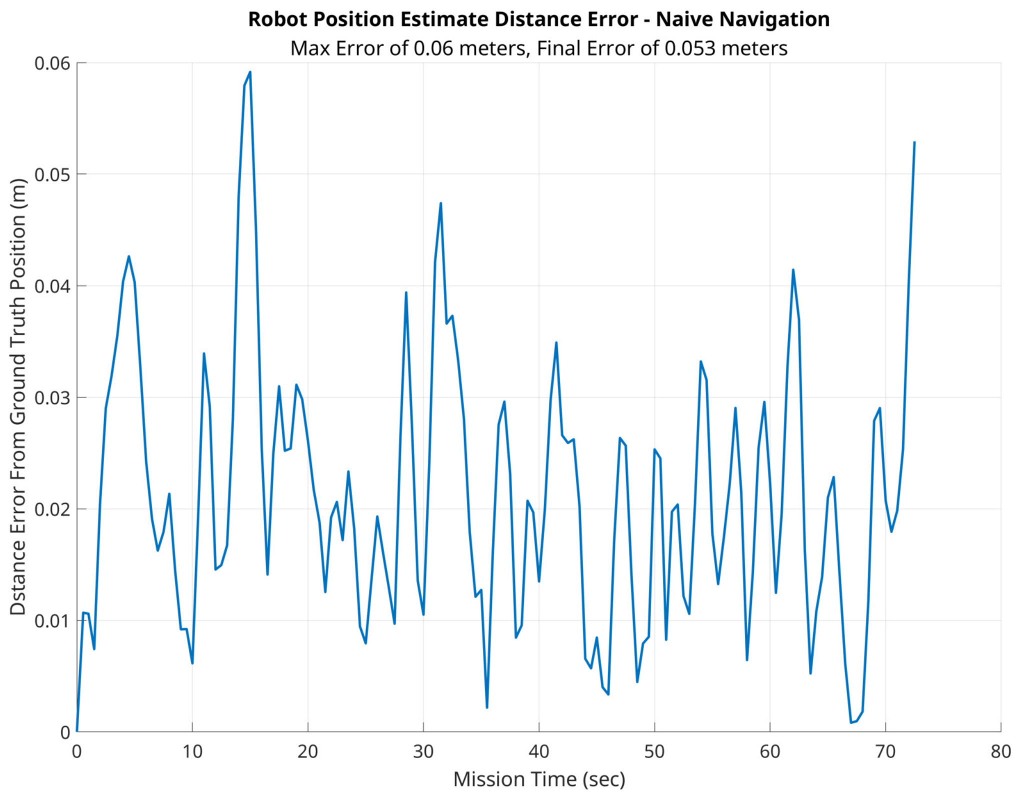

Figure 4.5: Naive Navigator State Estimate Error from Ground Truth

In contrast, the path planner intentionally sought out the nearest observable zone on the way to the goal in an attempt to maintain a low state estimate uncertainty. However, in this case, the observable zone was located far enough away that the robot was not be able to reliably achieve the desired path. The sequence of best paths selected during path planning looked promising and is shown in Figure 4.12, but sometimes the robot missed the first observable zone due to system noise and became lost as shown in Figure 4.13.

This comparison of simulation results was repeated 93 times to see the distribution of final error that could be expected in a real trial. Paths for both sets of simulations are overlaid in Figure 4.14 with the final robot positions shown in Figure 4.15. The covariances of final positions for each method of navigation were computed to draw 95% confidence ellipses indicating a modest improvement of

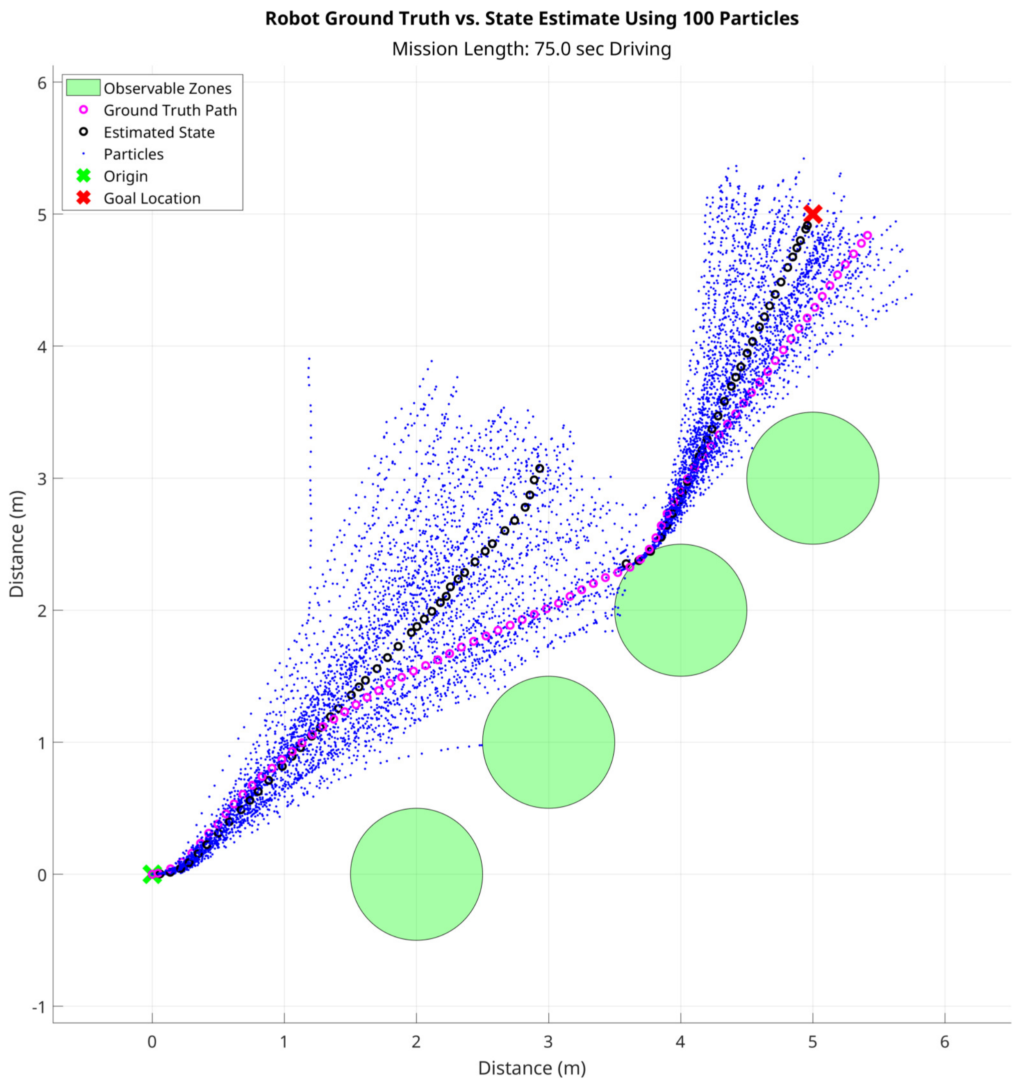

Figure 4.6: Naive Navigator Driving Past Observable Zones

the path planner over naive navigation in this scenario. Table 2 shows the state estimate error and time requirements for each approach.

4.2.4 Case Study #3

The third case study demonstrated the performance of the path planner when the nearest observable zone was beyond the time horizon that the path planner was set to explore. In this study, the goal was also located at (5,5) and two observable zones were positioned to either side of the direct path toward the goal, each with

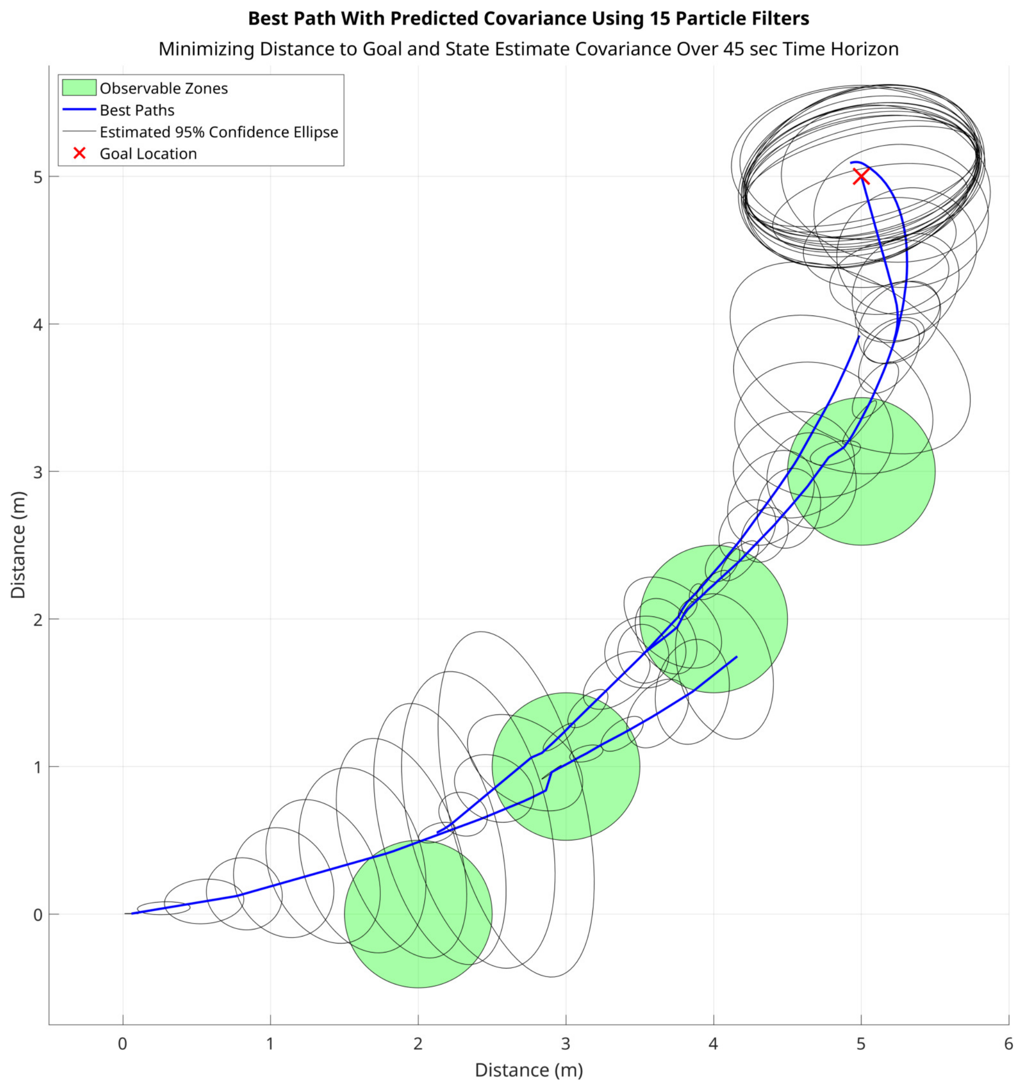

Figure 4.7: Path Planner Selected Paths Projected Through Observable Zones

a radius of 0.5 m and located at (3,5) and (5,3).

In the case of the naive controller heading directly for the the goal, the state estimate steadily diverged as expected, but the observable areas happened to be arranged to catch the robot if it strayed too far off track. An example of this behavior is shown in Figure 4.16.

Initially, the path planner pursued a blended exploratory and goal pursuit trajectory in the absence of observable zones, but once the robot was within the

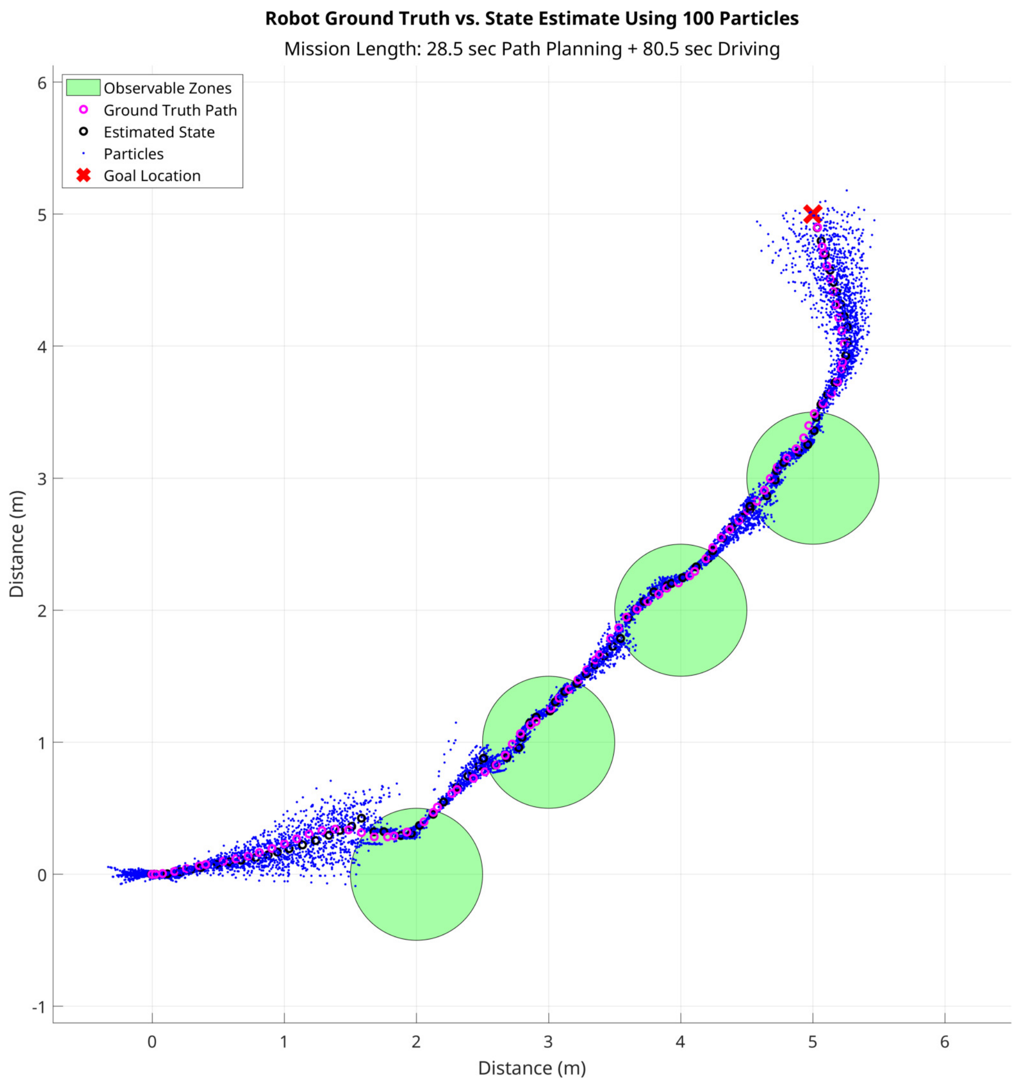

Figure 4.8: Path Planner Robot Navigating Through Observable Zones

time horizon from a mapped observable zone, the robot reoriented to intentionally seek out the observable zone. However, the state estimate was sometimes so poor at this point that the robot was not able to find the observable zone. A favorable sequence of best paths for this study is shown in Figure 4.17, but the ground truth in Figure 4.18 shows the reality that the robot missed the observable area and could not find the goal.

This comparison of simulation results was repeated 105 times to see the dis-

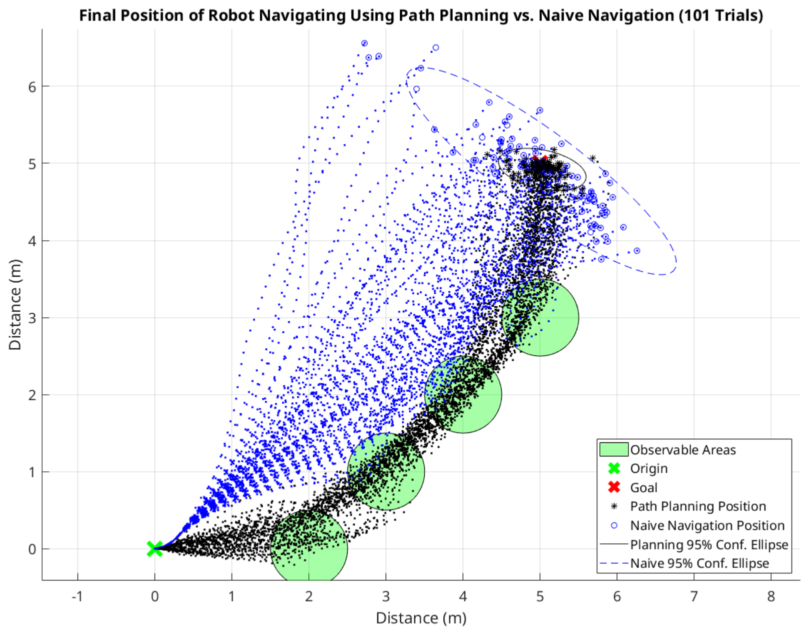

Figure 4.9: Path Comparison of Path Planning vs Naive Navigation (101 Trials)

Statistic	Calculation Method	Naive Navigator	Path Planner
Position Estimate Error en Route (m)	Absolute Mean Value	0.419	0.326
	Root Mean Square	0.789	0.568
Final Position Error (m)	Absolute Mean Value	1.037	0.614
	Root Mean Square	1.457	0.955
Initialization Time (s)	Absolute Mean Value	9.957	45.46
	Root Mean Square	11.44	47.95
Drive Time (s)	Absolute Mean Value	73.74	87.72
	Root Mean Square	73.78	88.14

Table 2: Naive Navigation vs. Path Planner Statistics for Case #2 (93 Trials)

tribution of final error that could be expected in a real trial. Paths for both sets of simulations are overlaid in Figure 4.19 with the final robot positions shown in Figure 4.20. The covariances of final positions for each method of navigation were computed to draw 95% confidence ellipses indicating that the path planner produces similarly poor results to naive navigation in this scenario. Table 3 shows the

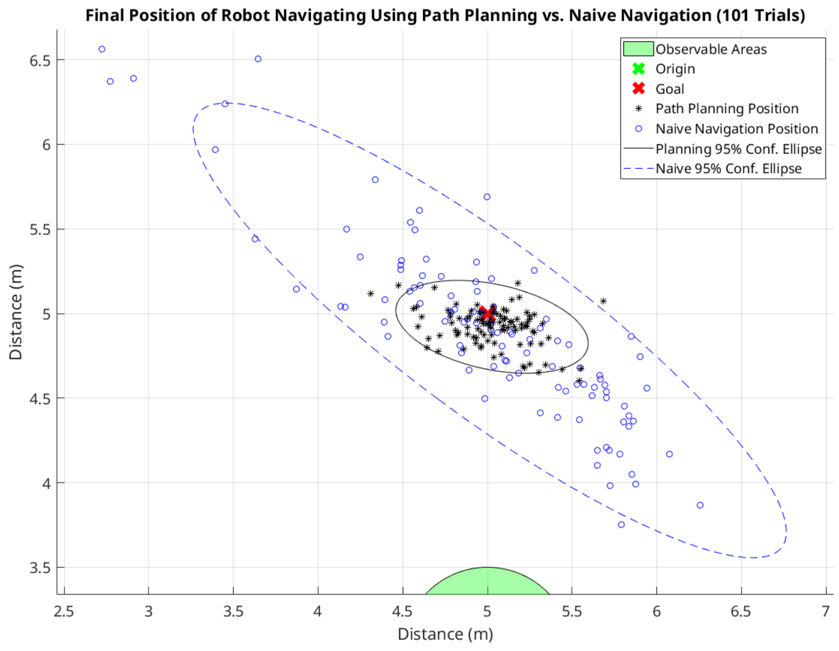

Figure 4.10: Final Position Comparison of Path Planning vs Naive Navigation (101 Trials)

state estimate error and time requirements for each approach.

Statistic	Calculation Method	Naive Navigator	Path Planner
Position Estimate Error en Route (m)	Absolute Mean Value	0.530	0.385
	Root Mean Square	1.015	0.679
Final Position Error (m)	Absolute Mean Value	1.158	0.858
	Root Mean Square	1.858	1.156
Initialization Time (s)	Absolute Mean Value	1.648	16.62
	Root Mean Square	1.665	16.72
Drive Time (s)	Absolute Mean Value	74.24	91.49
	Root Mean Square	74.29	91.93

Table 3: Naive Navigation vs. Path Planner Statistics for Case #3 (105 Trials)

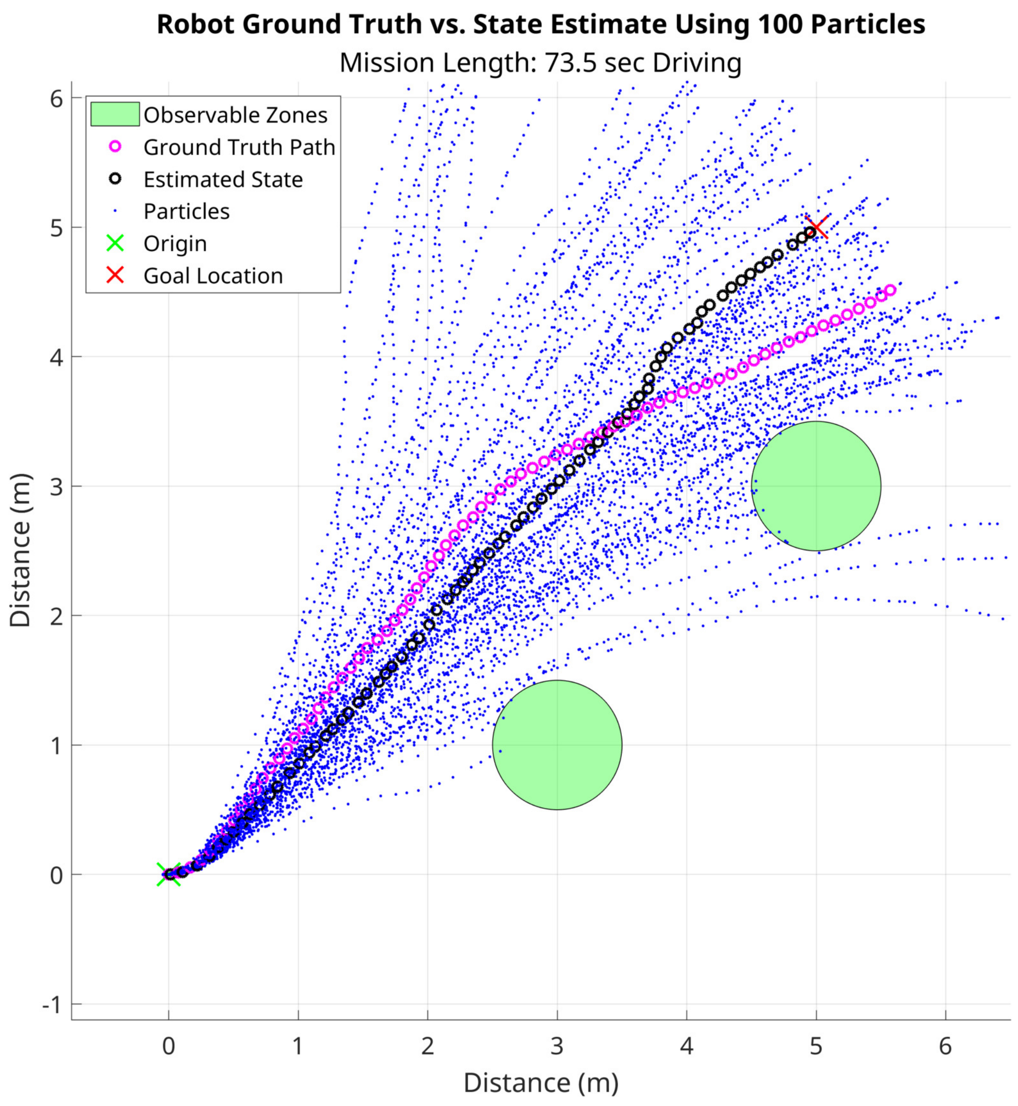

Figure 4.11: Naive Navigator Driving Past Observable Zones

4.2.5 Case Study Randomly Generated

The final case study demonstrated a positive outcome in a randomly generated scenario with approximately 30.7% beacon coverage in the simulated environment. In this case the goal was located at about (-8.03, 5.96) and there was an observability void between the origin and goal positions, but the void was surrounded by beacons.

In the case of the naive controller heading directly for the the goal, the state

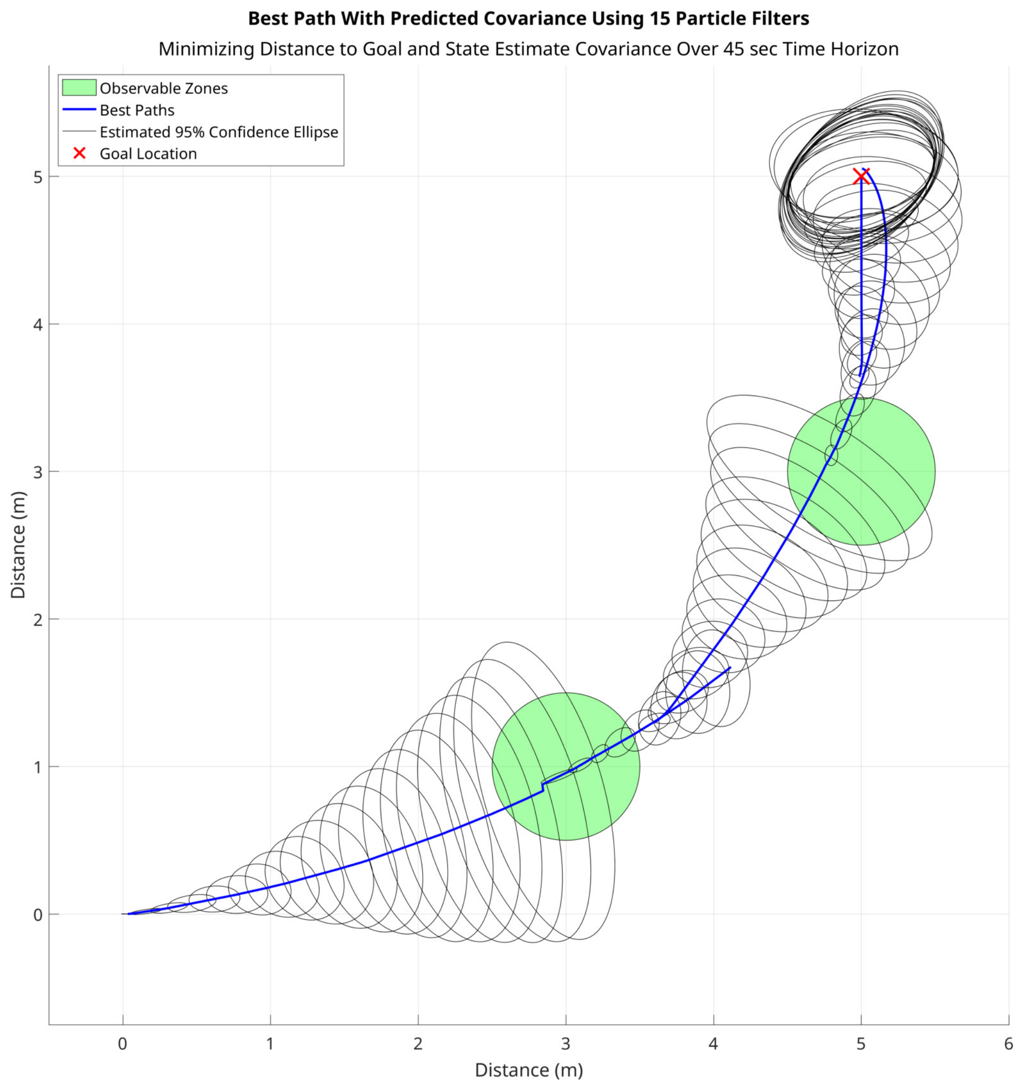

Figure 4.12: Path Planner Selected Paths Projected Through Observable Zones

estimate confidence steadily diverged until the robot reached the other side of the void and encountered an observable region, shown in Figure 4.21.

In contrast, the path planner explored the potential paths shown in Figure 4.22 and intentionally traveled around the perimeter of the observability void to maintain a low state estimate uncertainty as it progressed towards the goal. The sequence of best paths selected is shown in Figure 4.23 with the resulting state estimate shown in Figure 4.24.

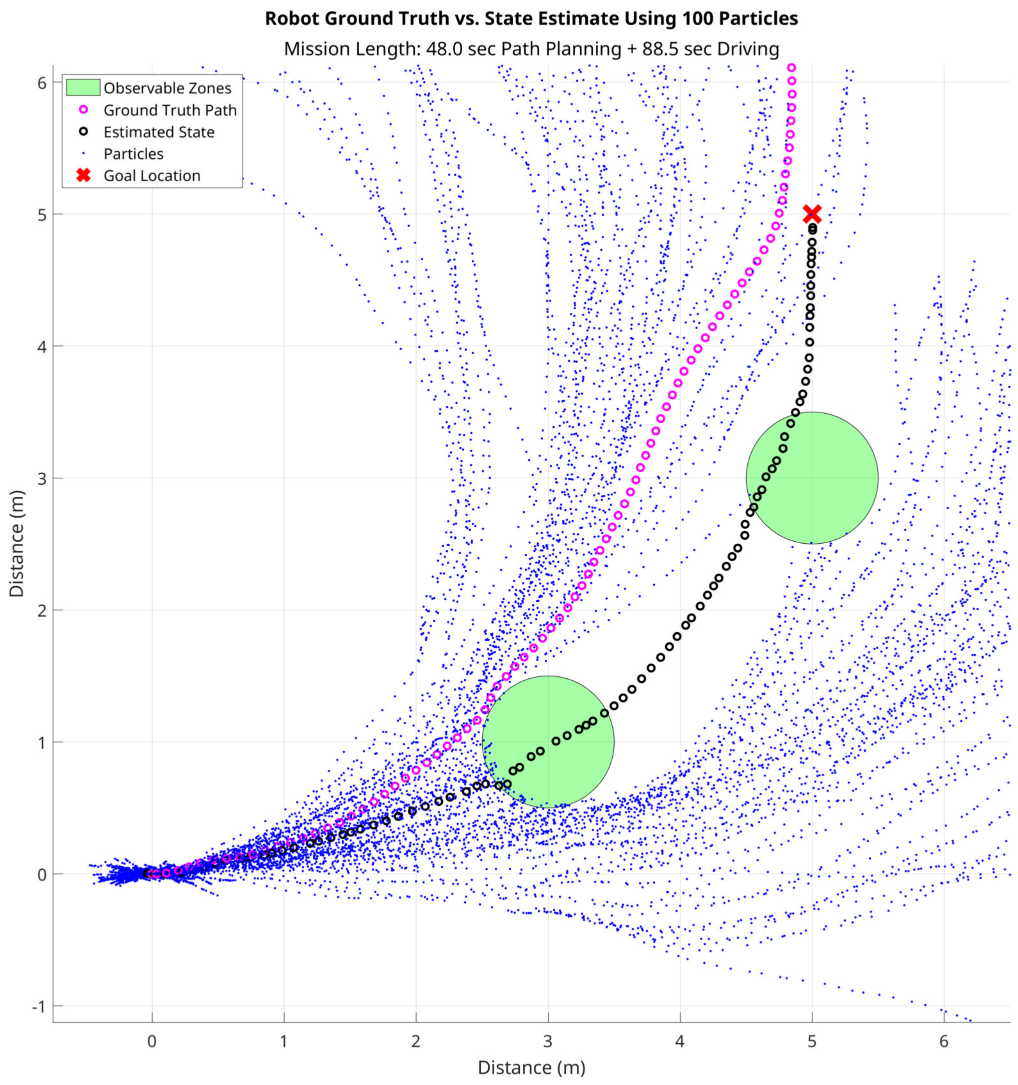

Figure 4.13: Path Planner Robot Missing the Observable Zones

The euclidean distance between the state estimate and ground truth at each time interval were plotted for the path planner and naive navigation for this random case. Even though the final naive navigation state estimate happened to be favorable in this trial, the path planner maintained a lower maximum position estimate error of 0.16 m compared to the naive navigation maximum position estimate error of 0.32 m as shown in Figure 4.25 vs Figure 4.26 respectively.

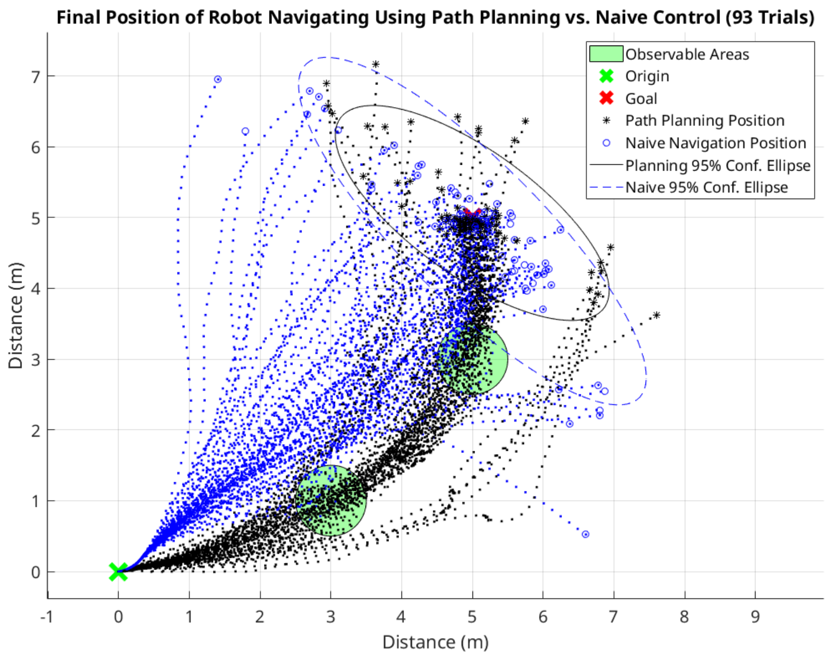

Figure 4.14: Case #2 Path Comparison of Path Planning vs Naive Navigation (93 Trials)

4.3 Random Trials

Environments were randomly generated to assess the performance of the path planning system compared to the standard naive navigation. After each system reached what it believed to be the goal position, the state estimate logs, ground truth logs, goal position, and map characteristics were saved for post processing. Beacons for these trials had a detection radius of 1 m and were arbitrarily placed in the environment to provide a coverage ratio of between 1% and 50%. More trials were selected for lower coverage concentrations due to the higher variability of outcomes, with the final distribution of environment beacon coverage shown as a histogram in Figure 4.27. A new goal located 10 m from the origin was also generated for each comparison trial with a distribution from the origin shown in

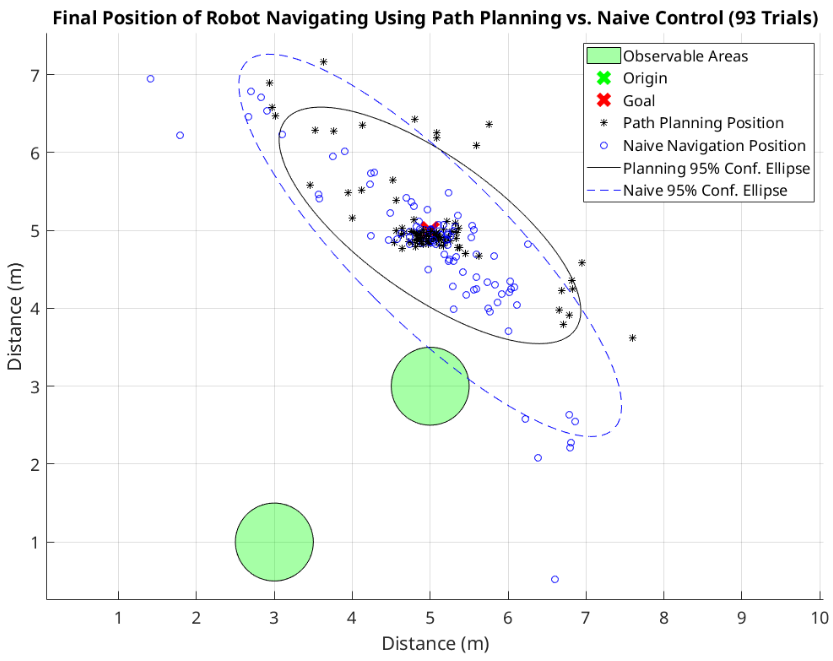

Figure 4.15: Case #2 Final Position Comparison of Path Planning vs Naive Navigation (93 Trials)

Figure 4.28.

In all, 1,818 randomly generated environments were evaluated to compare the performance of the path planner against the naive navigation system. For each trial, the mean absolute position error en route to the goal with respect to the ground truth was calculated to quantify the performance of each system during its respective trial. These values are plotted in Figure 4.29 along with 2^{nd} order least-squares regression functions to characterize the relative performance of these approaches across many environments. These data suggest that the path planner improves en route state estimation compared to naive navigation in environments with between 1% and 50% beacon coverage in an otherwise unobservable environment. However, the improvement came at the cost of extra time required to pursue

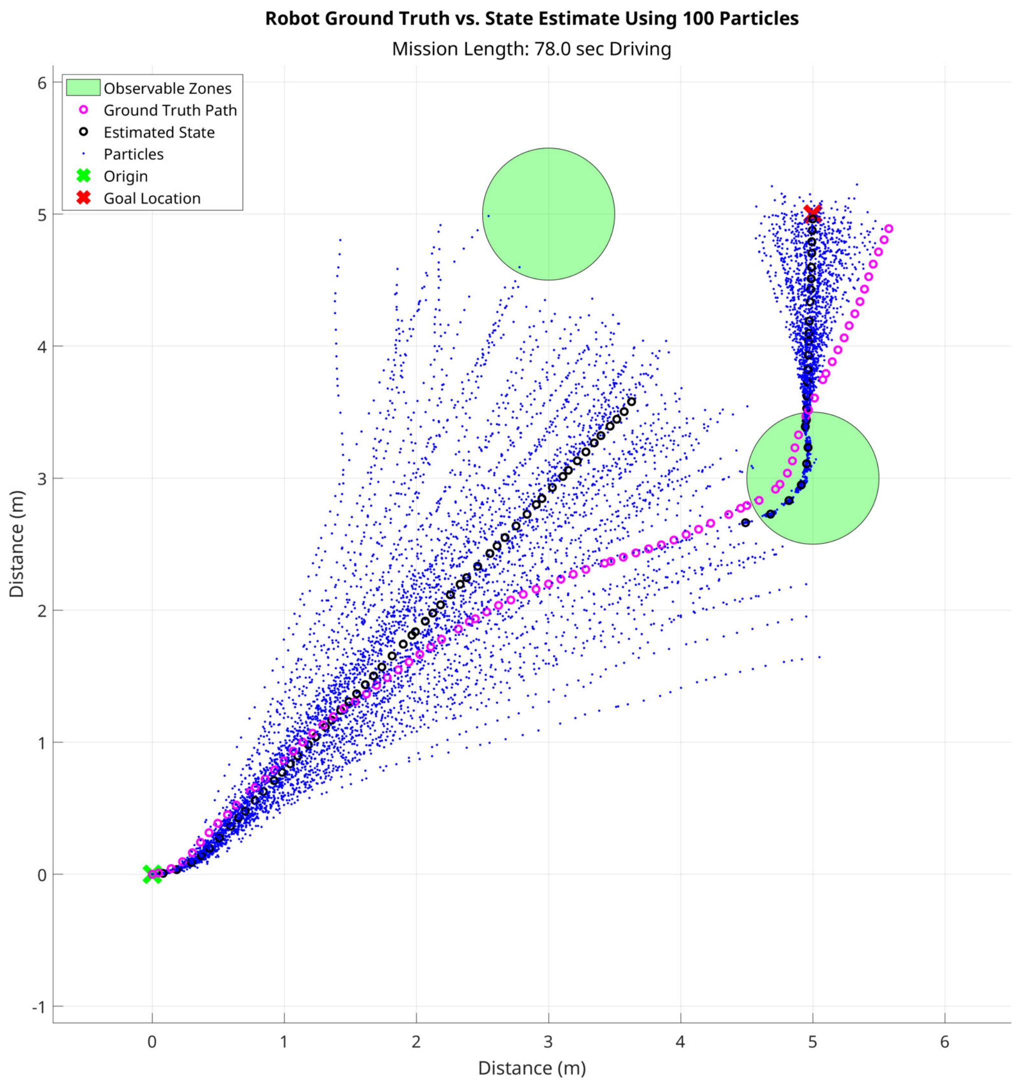

Figure 4.16: Naive Navigator Driving Between Observable Zones

observable zones with longer drive times required to reach what the robot believed was the goal location. This trade-off is shown in Figure 4.30 along with 2^{nd} order least-squares regression functions to characterize the relative performance of these approaches across many environments. The regression functions were calculate best fit distance error and drive time values shown in Table 4 the compare the path planning and naive navigation outcomes for environments with varying beacon coverage.

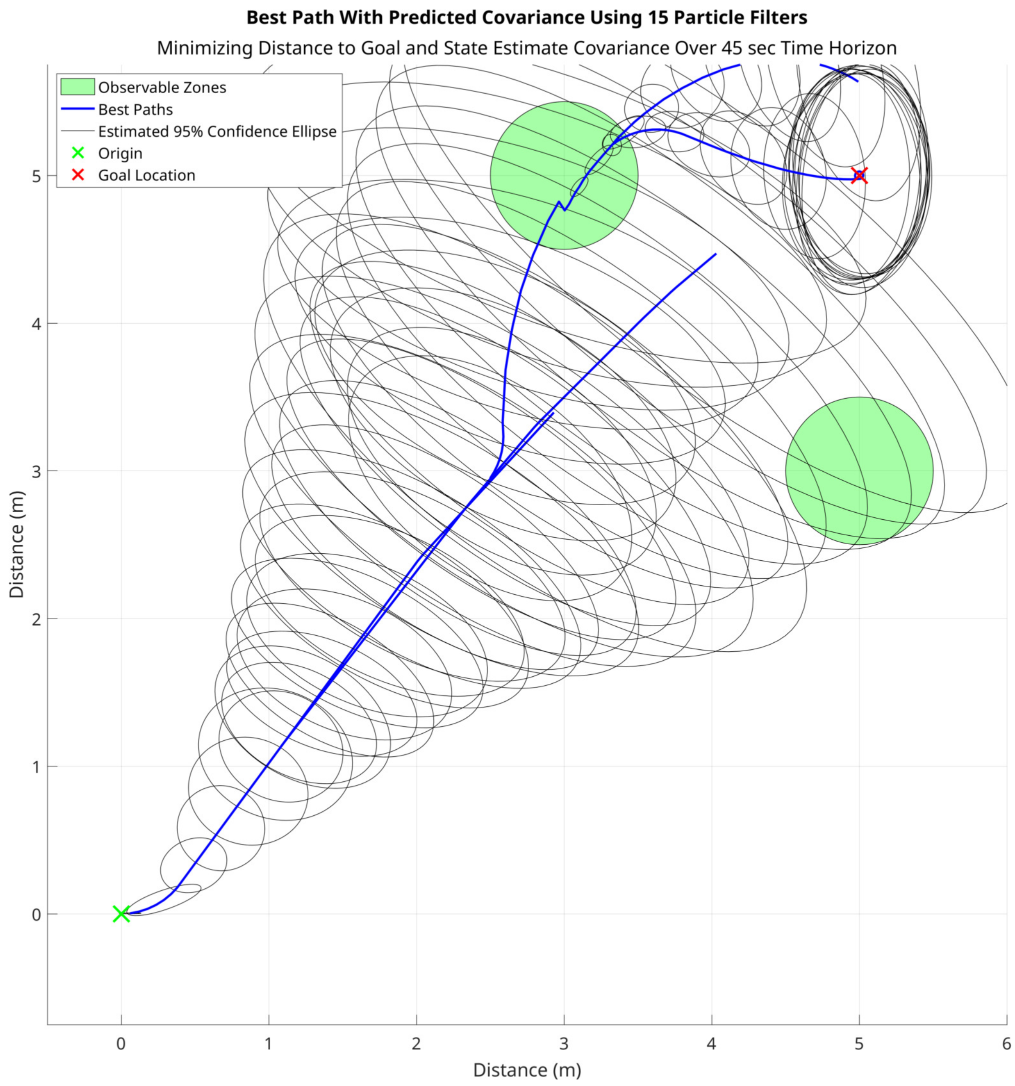

Figure 4.17: Path Planner Selected Paths Projected Through Observable Zones

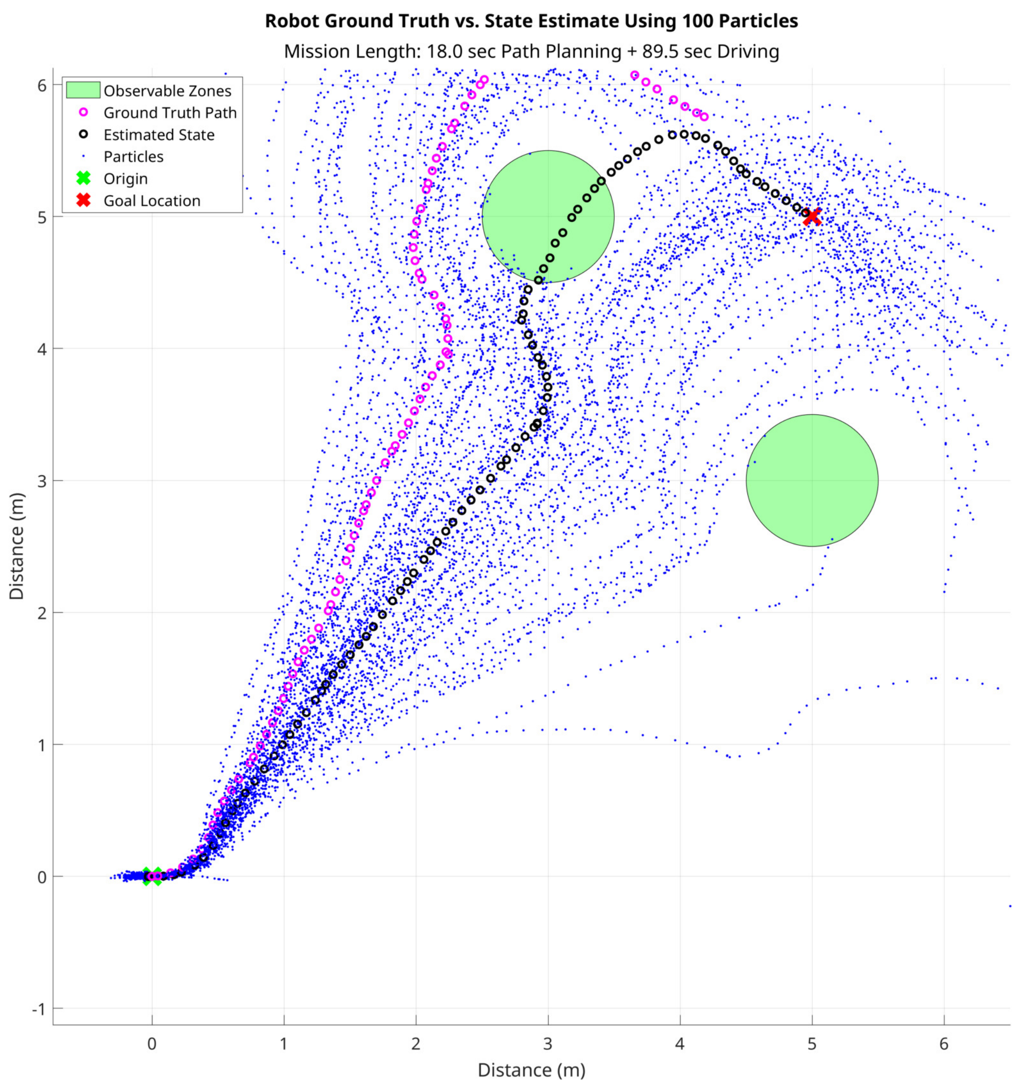

Figure 4.18: Path Planner Robot Missing the Observable Zones

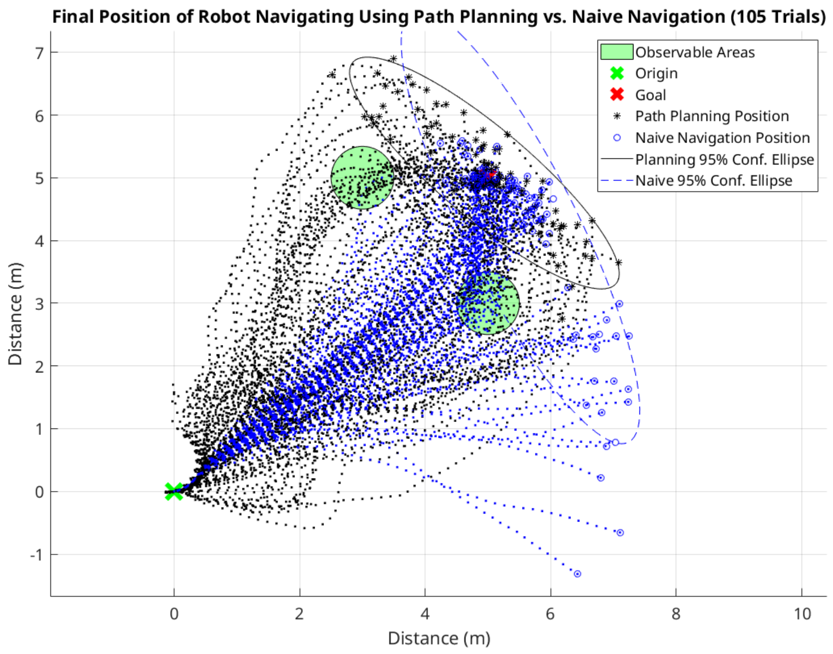

Figure 4.19: Case #3 Path Comparison of Path Planning vs Naive Navigation (105 Trials)

Performance Metric	Beacon Coverage	Naive Navigator	Path Planner
Position Estimate Error en Route (m)	1%	0.759	0.508
	10%	0.548	0.326
	20%	0.364	0.177
	30%	0.233	0.085
	40%	0.155	0.048
	50%	0.131	0.068
Drive Time (sec)	1%	110.6	140.2
	10%	112.1	136.7
	20%	112.4	132.3
	30%	111.1	127.1
	40%	108.2	121.3
	50%	103.8	114.9

Table 4: Naive Navigation vs. Path Planner Performance Statistics (1818 Randomly Generated Trials)

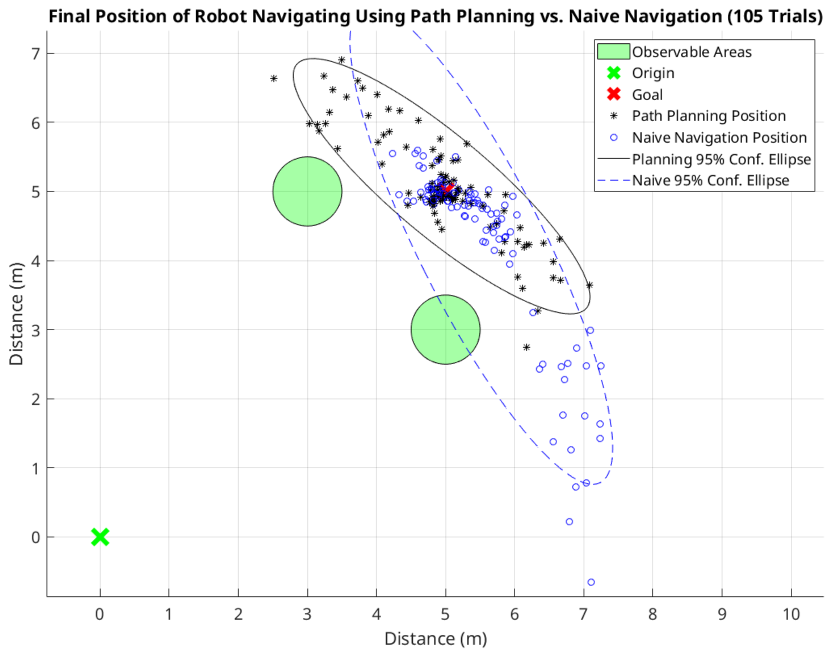

Figure 4.20: Case #3 Final Position Comparison of Path Planning vs Naive Navigation (105 Trials)

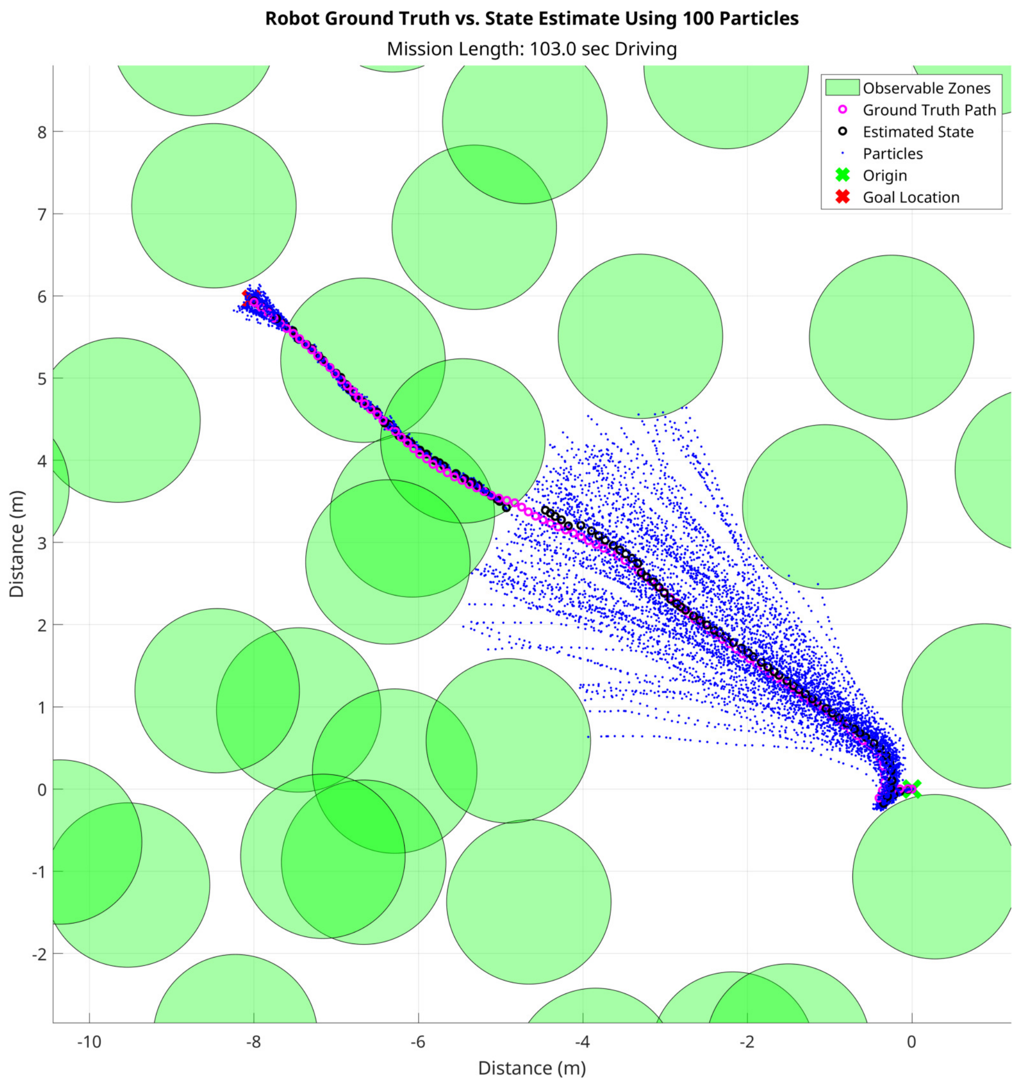

Figure 4.21: Naive Navigator Driving Through a Randomly Constructed Environment

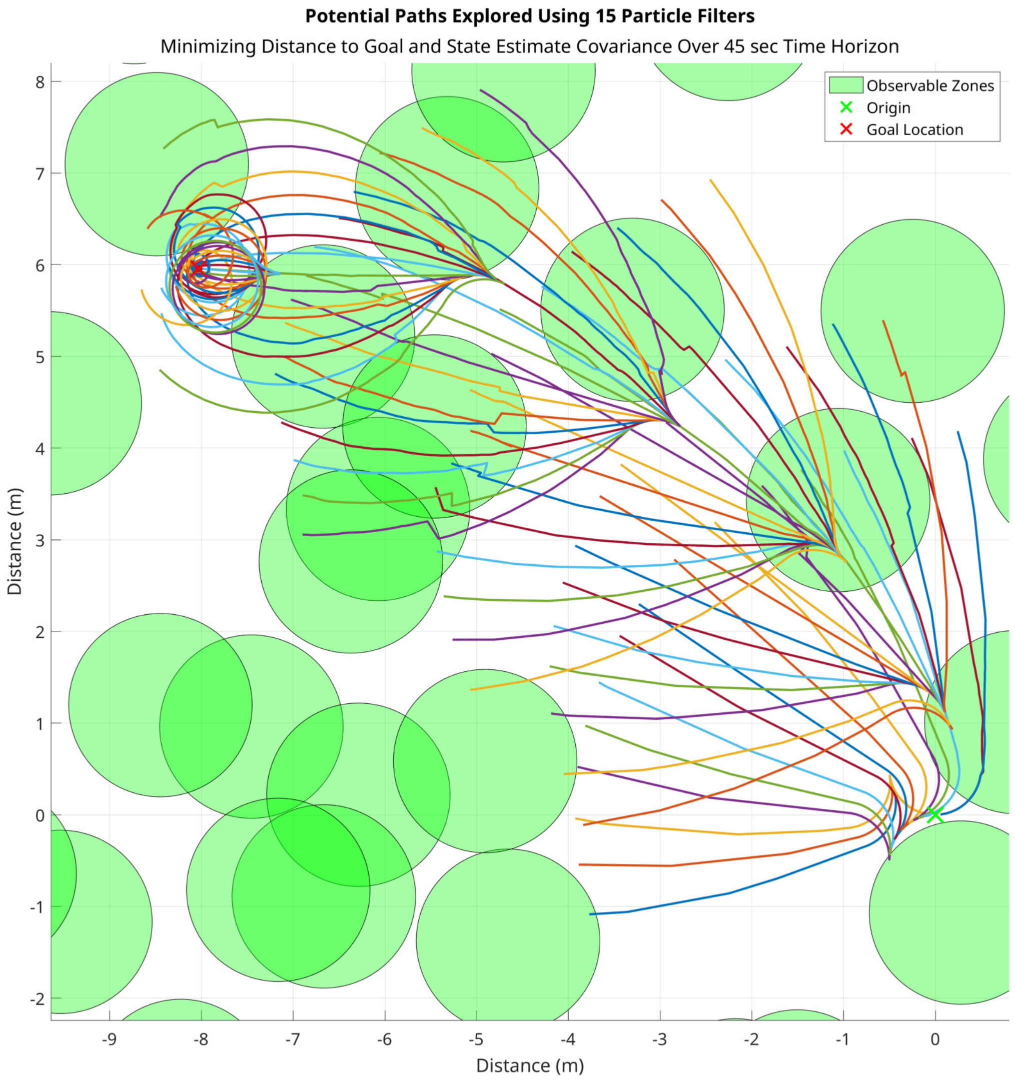

Figure 4.22: Potential Paths Evaluated to Circumnavigate an Observability Void

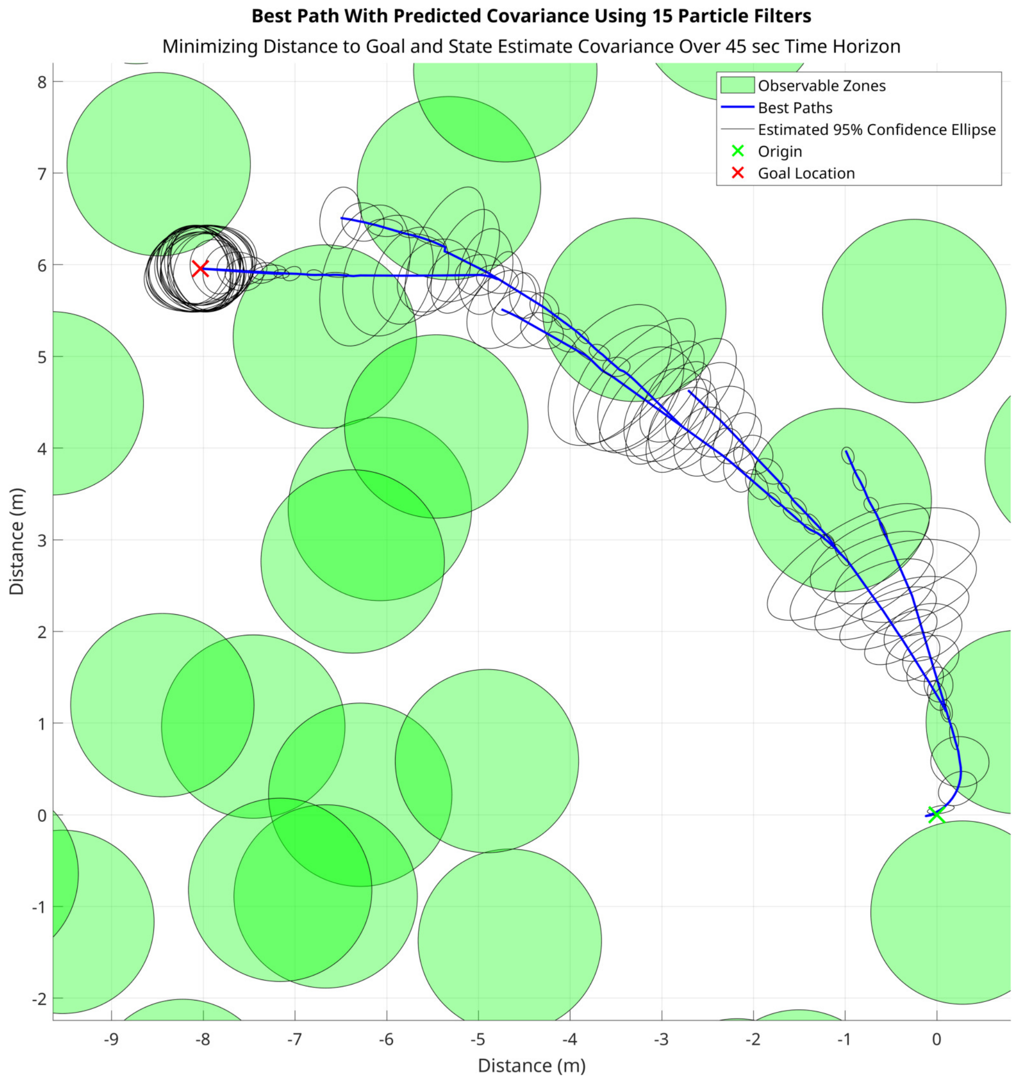

Figure 4.23: Path Planner Selected Paths Circumnavigating an Observability Void

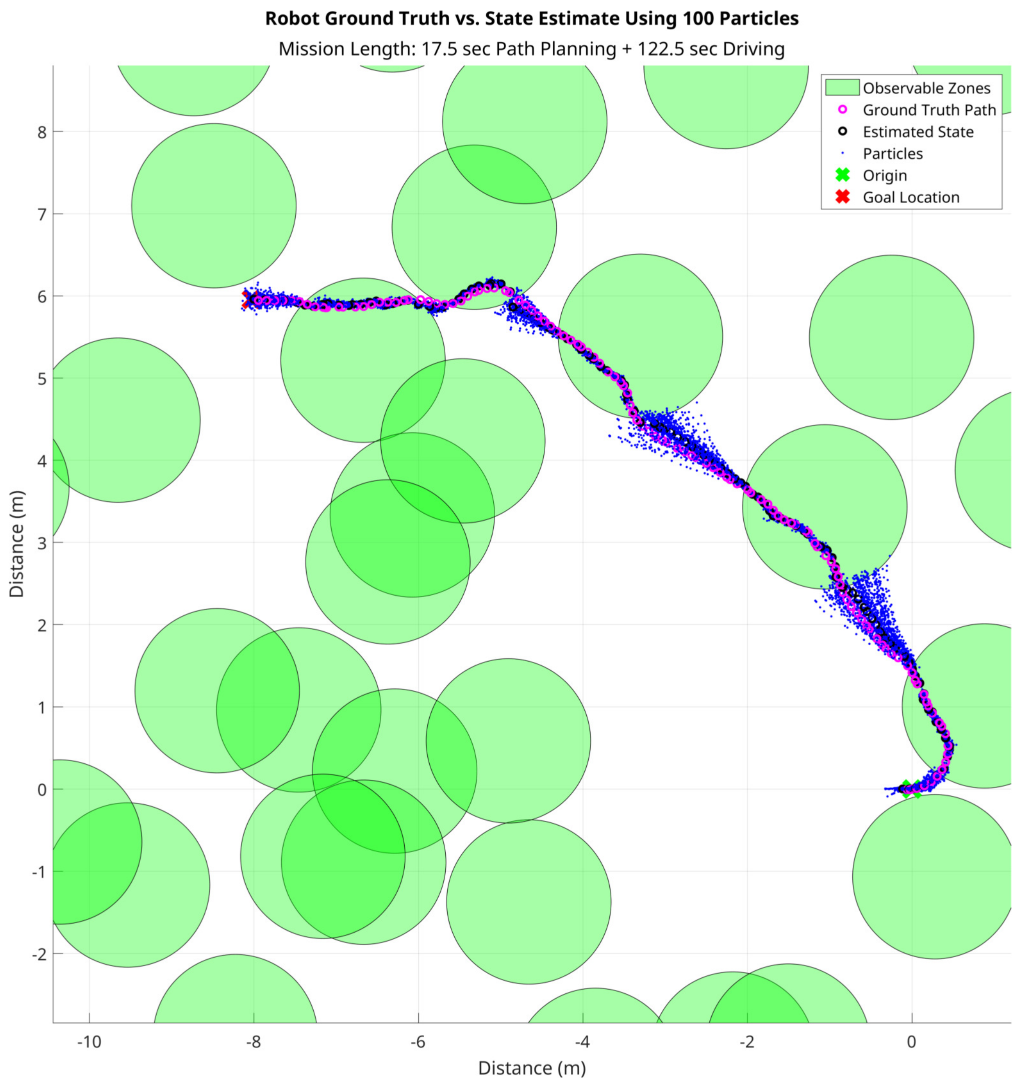

Figure 4.24: Path Planner Robot State Estimate Circumnavigating an Observability Void

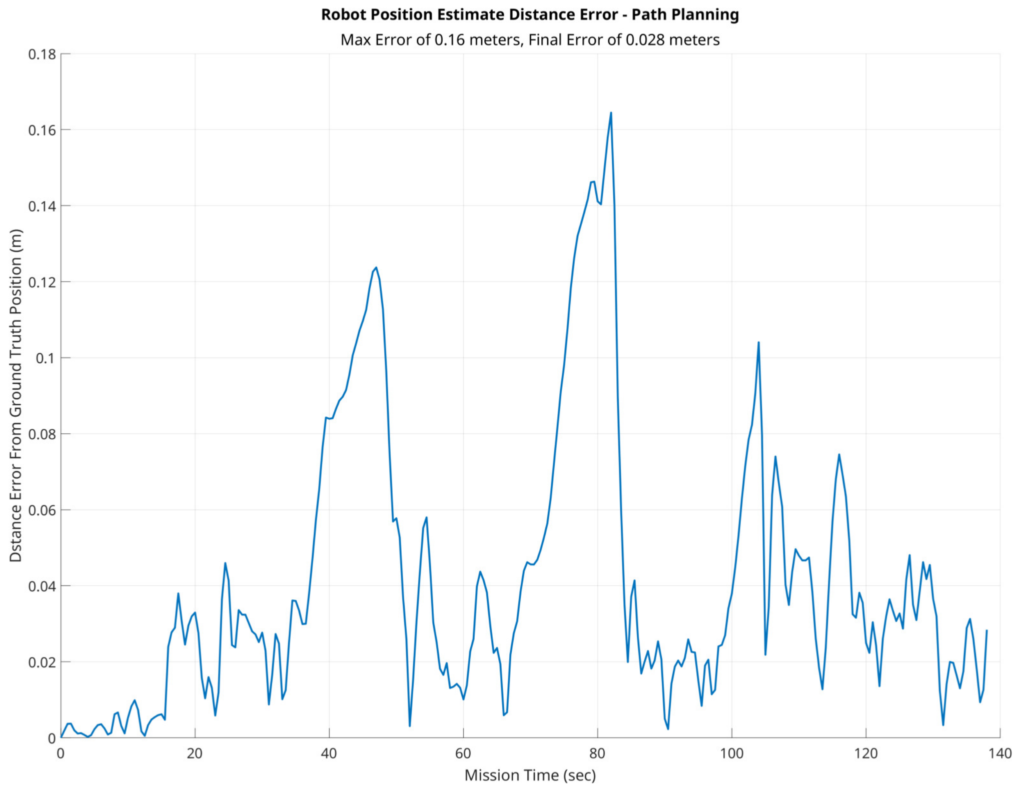

Figure 4.25: Path Planner Position Estimate Error from Ground Truth

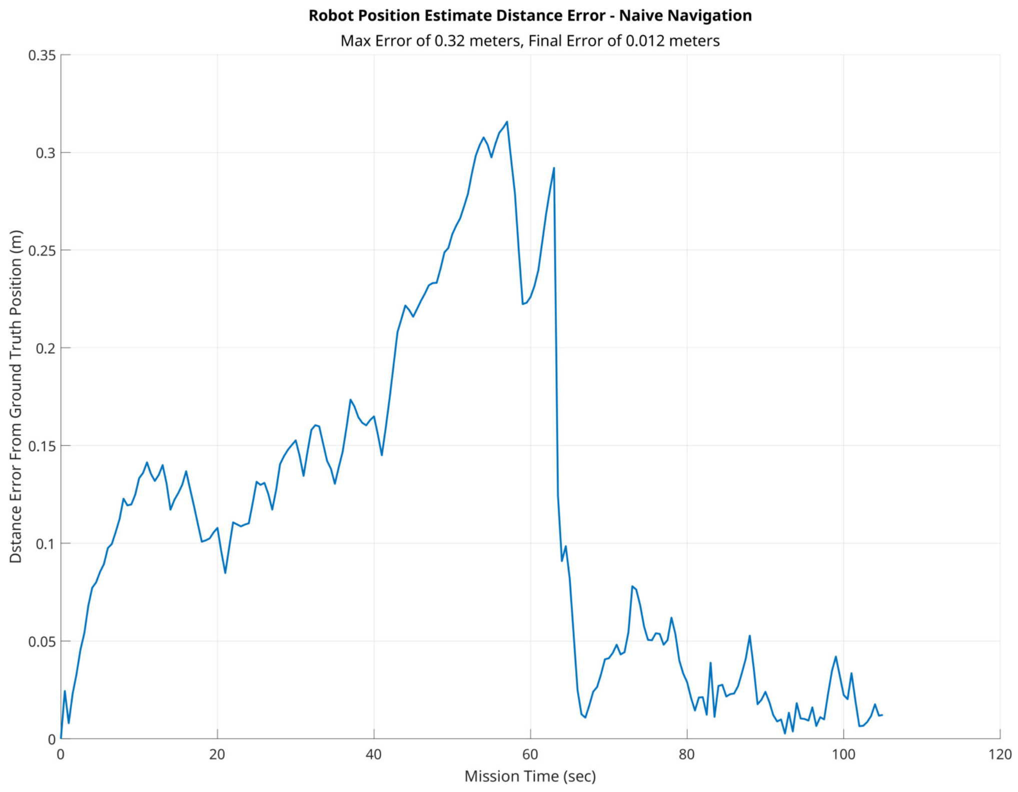

Figure 4.26: Naive Navigation Position Estimate Error from Ground Truth

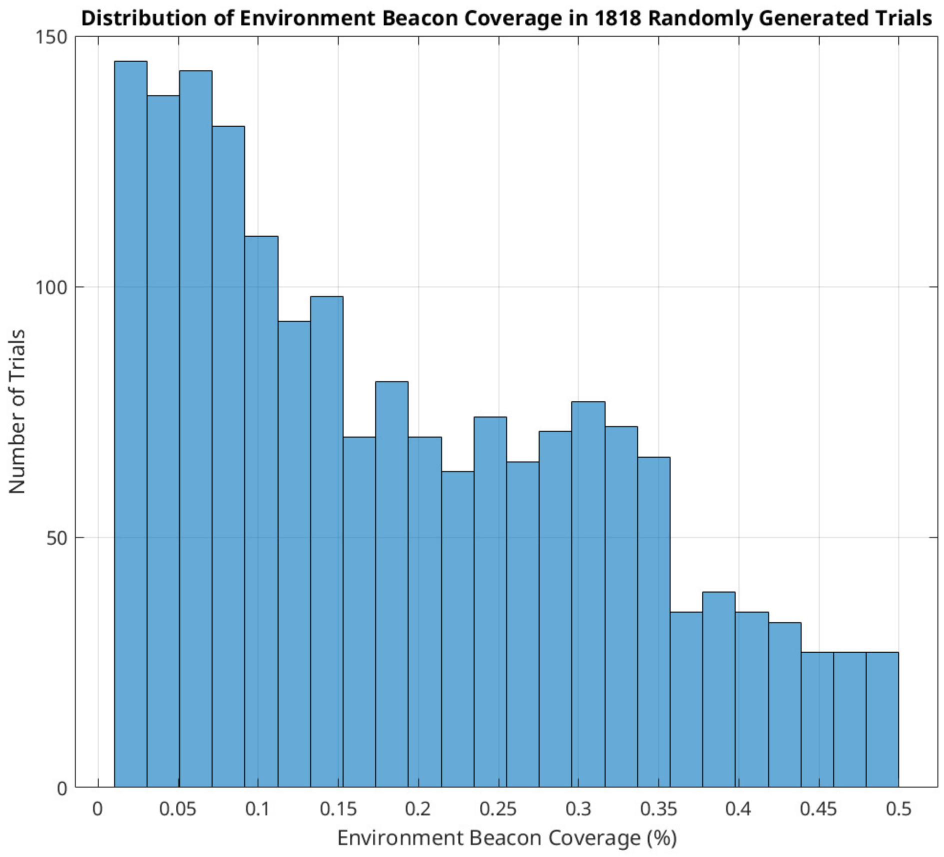

Figure 4.27: Distribution of Environment Beacon Coverage in Randomly Generated Trials

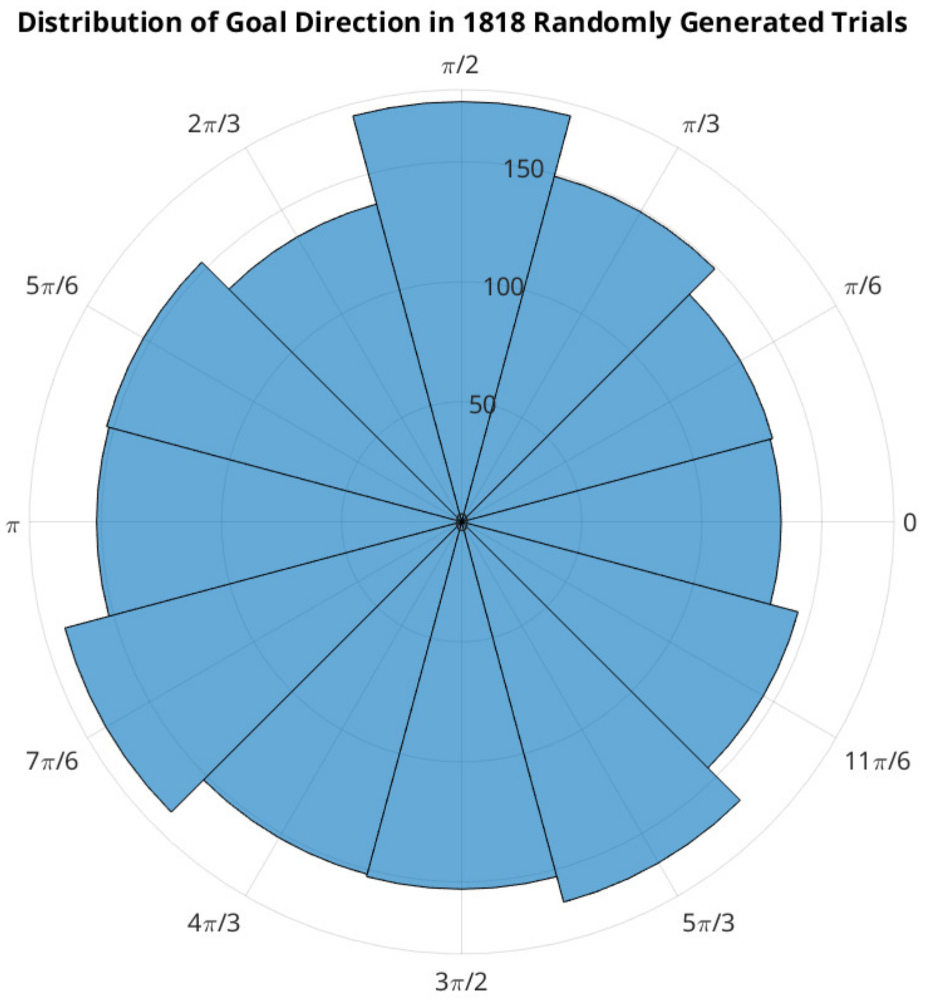

Figure 4.28: Distribution of Goal Direction in Randomly Generated Trials

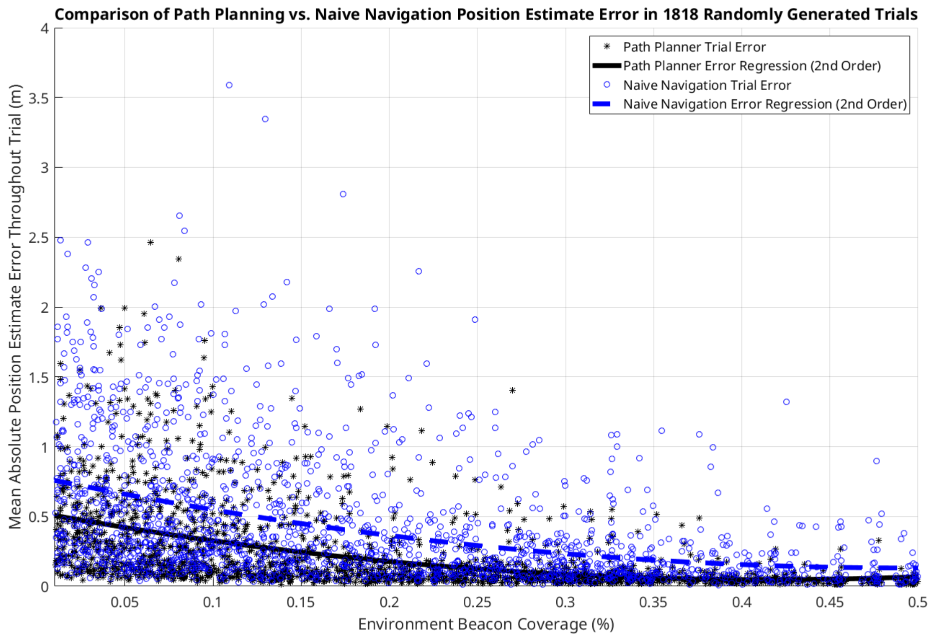

Figure 4.29: Comparison of Path Planning vs. Naive Navigation Position Estimate Error in Randomly Generated Trials

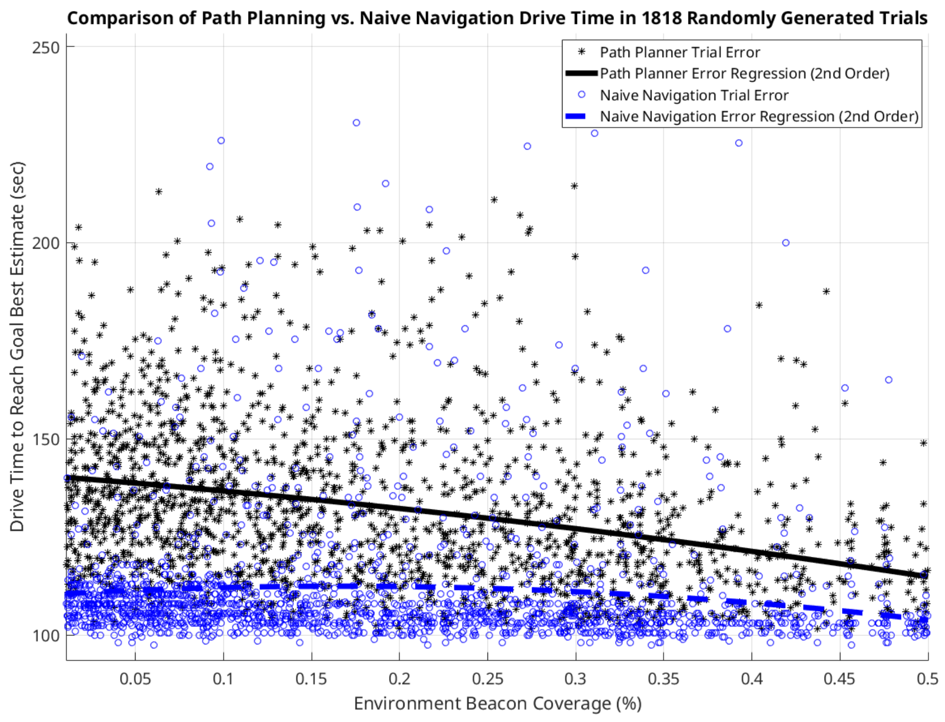

Figure 4.30: Comparison of Path Planning vs. Naive Navigation Drive Time in Randomly Generated Trials

CHAPTER 5

Experiments

To validate the simulation results achieved in Chapter 4, Robot Operating System (ROS) launch files were developed to integrate the mobile ground robot as a source of odometry feedback and the OptiTrack Motion Capture system as the measurement source. The new system configuration is shown in Figure 5.1.

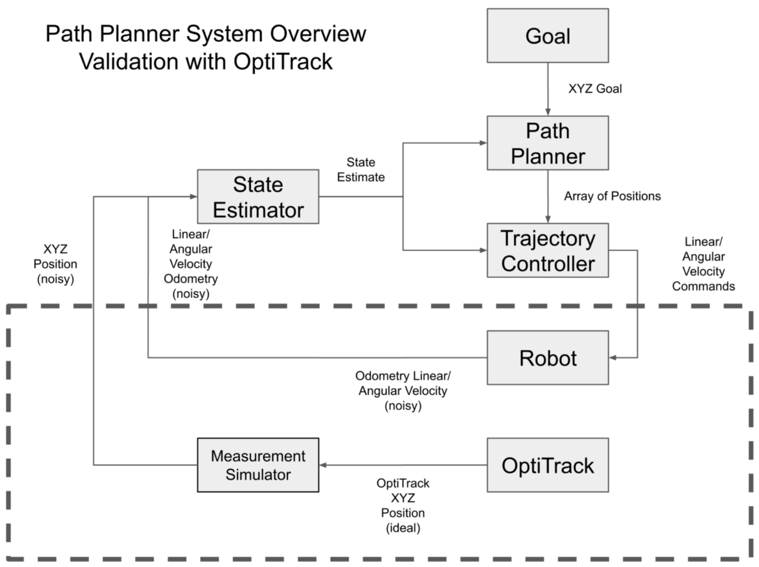
Path Planner System Overview
Validation with OptiTrack
Goal
XYZ Goal
Path Planner
State Estimate
State Estimator
Array of Positions
Trajectory Controller
Linear/ Angular Velocity Commands
XYZ Position (noisy)
Linear/ Angular Velocity Odometry (noisy)
Robot
Odometry Linear/ Angular Velocity (noisy)
Measurement Simulator
OptiTrack
OptiTrack XYZ Position (ideal)

Figure 5.1: System Configuration Overview - OptiTrack Validation

5.1 Incorporating the Ground Robot

As a ROS enabled robot, the mobile ground robot communication was straightforward to configure for use in this research. Instead of launching the Gazebo simulator and noisy odometry node that was previously used in simulation, the mobile ground robot was energized with a battery pack and the on-board Odroid computer was configured to communicate with the other ROS nodes on the local network. First, environment variables were set to indicate which machine on the local network was running the ROS core, then a ROS serial server node was started on the robot to parse information going to and from the Arduino motor control board. Finally, the ROS serial server registered with the remote ROS core to register routing paths for the ROS topics it published and subscribed to.

5.2 Incorporating OptiTrack

The OptiTrack Motion Capture System was not directly compatible with ROS, but instead produced a continuous stream of position and motion information on the network. A generic Virtual Reality Peripheral Network (VRPN) client from the ros-noetic-vrpn-client-ros Ubuntu package converted the OptiTrack stream to a usable ROS topic. The aforementioned ROS launch file was used to spawn the VRPN client node given the IP address of the machine running the OptiTrack software and generate a standard Pose message corresponding to the tracked rigid body "robot" object. This Pose message replaced the Gazebo simulator, but due to the high accuracy of the OptiTrack system, noise was still added as before. Measurements were marked as either "good" or "bad" depending on the observability of each measurement to indicate whether the state estimator could use the information.

5.3 Simulation Validation

Several experiments conducted in simulation were repeated using the real robot to validate our results. Results from these trials are presented and analyzed below.

5.3.1 Case of Full Observability

As before, the trivial case for this research must compare the performance of the path planner against naive navigation in an environment with global full observability to ensure there was no loss in standard functionality. As expected, the path planner explored many paths shown in Figure 5.2, but ultimately selected the path directly leading to the goal, shown by the state estimates and ground truth in Figure 5.3.

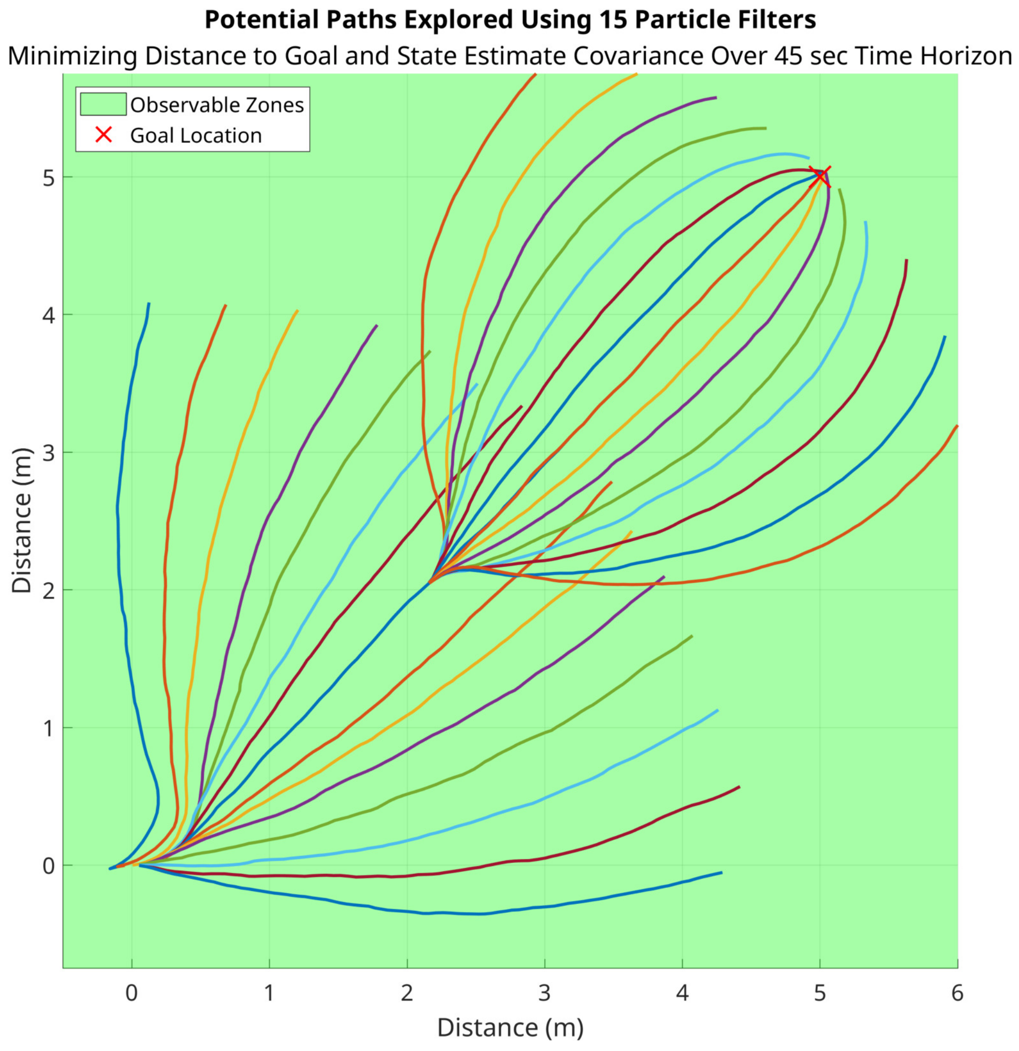

Figure 5.2: Global Full Observability Path Planner Performance

5.3.2 Case Study #1

As before, the first case study demonstrated the preference for traveling in an observable zone due to the resulting higher state estimate confidence. In this

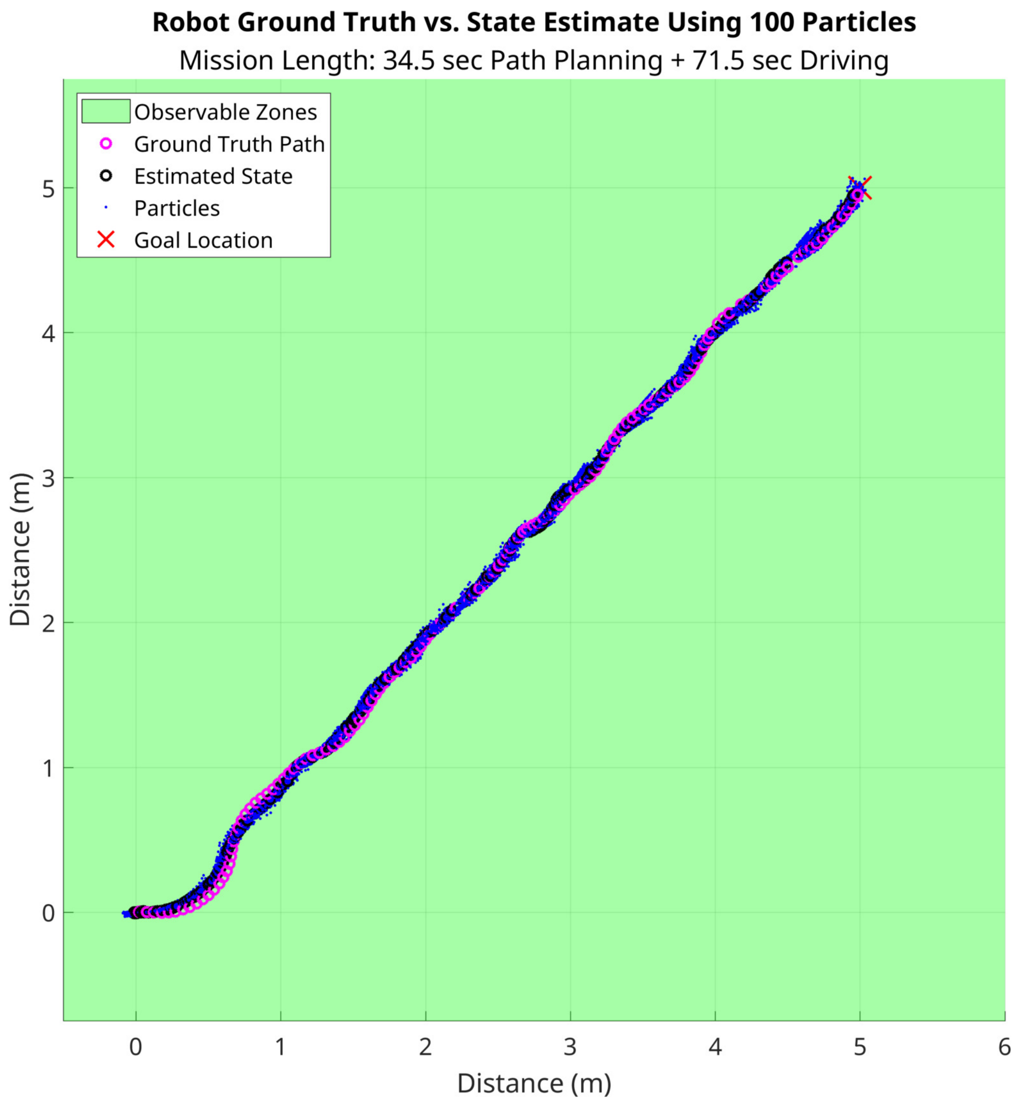

Figure 5.3: Global Full Observability Naive Navigation Performance

study, the goal was located at (5,5) and a row of observable zones was positioned to the side of the direct path to the goal, each with a radius of 0.5 m and located at (2,0), (3,1), (4,2), and (5,3).

In the case of the naive controller heading directly for the the goal, the state estimate steadily diverged unless the robot accidentally encountered an observable zone and the state estimate converged. This circumstance is shown in in Figure 5.4.

By contrast, the path planner intentionally sought out observable zones on the

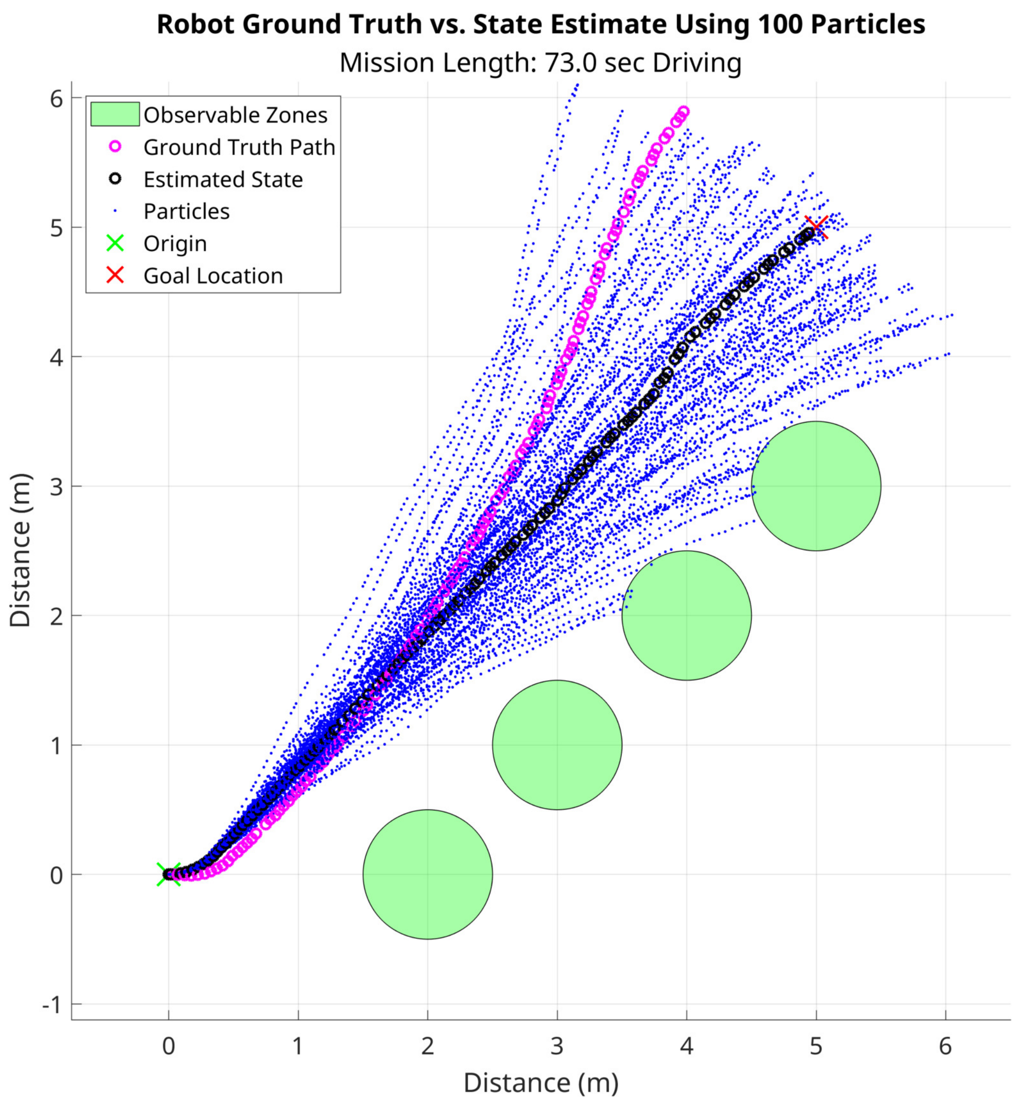

Figure 5.4: Naive Controller State Estimate Navigating Past Observable Zones

way to the goal to maintain a low state estimate uncertainty. The sequence of best paths selected during the path planning for this real trial is shown in Figure 5.5 with the resultant path and state estimate shown in Figure 5.6.

These results align nicely with the expected outcome and demonstrate that the simulations accurately predicted better performance from the path planner compared to the naive navigation. A comparison of the continuous state estimate error for the path planner and naive navigation trials is shown in Figure 5.7

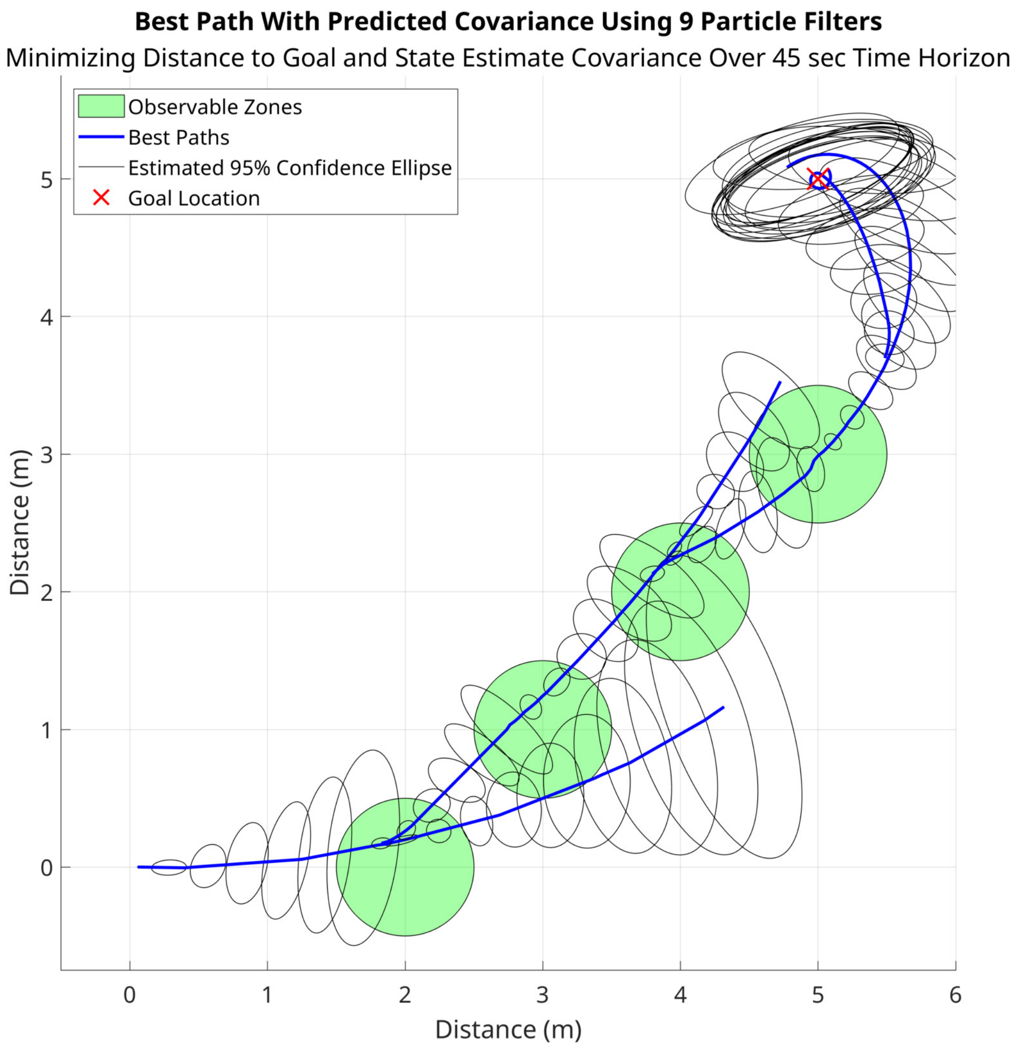

Figure 5.5: Path Planner Selected Paths Projected Through Observable Zones

and Figure 5.8 respectively. The performance of the path planner maintains the state estimate position error below 0.32 m while the naive navigation case shows unbounded error growth, reaching 1.35 m by the end of the trial.

5.3.3 Case Study #2

Again, the second case study demonstrated the limitations of the path planner when the environment was so sparsely populated with observable zones that the robot could not navigate as desired. In this study, the goal was also located at

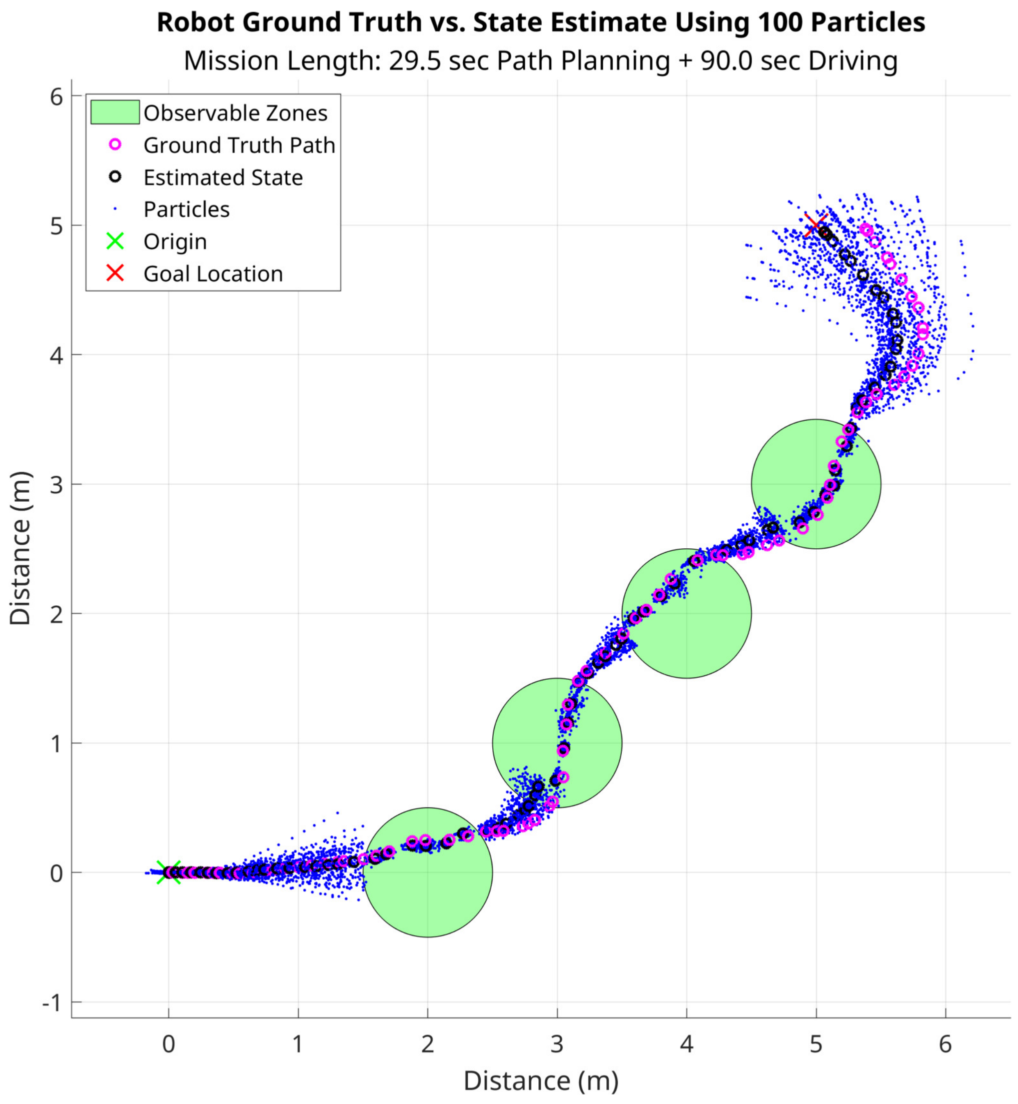

Figure 5.6: Path Planner State Estimate Navigating Through Observable Zones

(5,5) and two observable zones were positioned to the side of the direct path to the goal, each with a radius of 0.5 m and located at (3,1) and (5,3).

In the case of the naive controller heading directly for the the goal, the state estimate confidence again steadily increased throughout the trial. In this case, the robot actually benefited from pruning particles incorrectly propagated into an observable zone, and completed the trial reasonably close to the goal as shown in Figure 5.9.

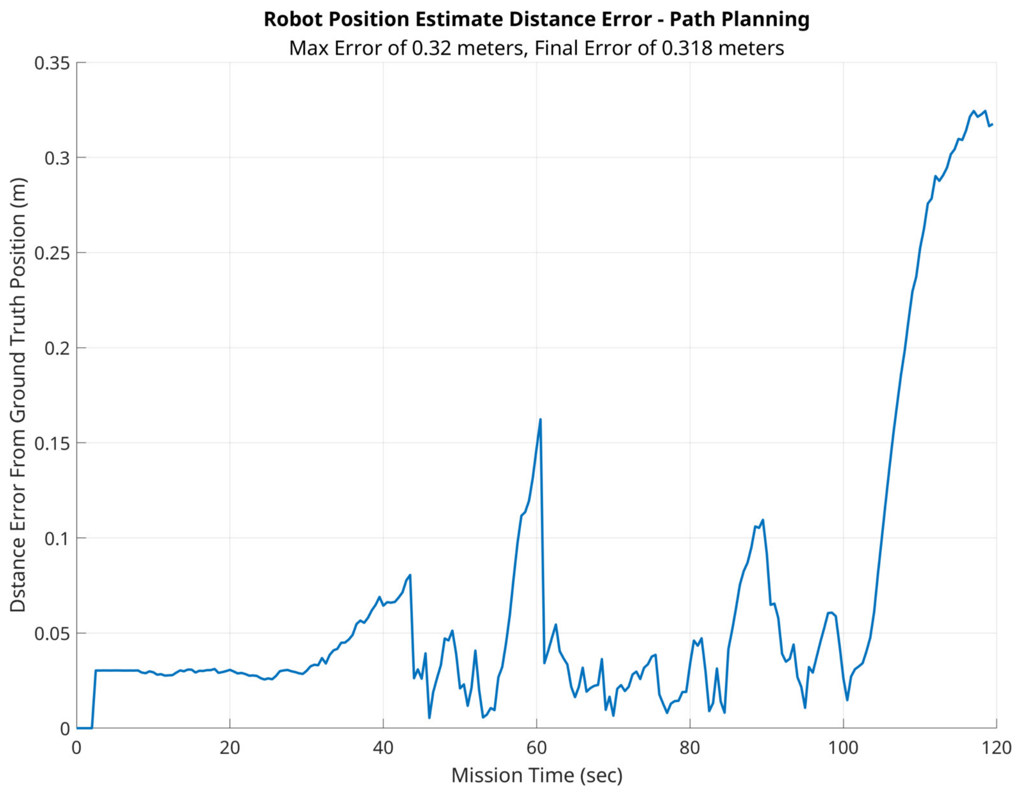

Figure 5.7: Path Planner State Estimate Error from Ground Truth

In contrast, the path planner intentionally sought out the nearest observable zone on the way to the goal in an attempt to maintain a low state estimate uncertainty. In this case, the robot was able to find the observable zone and converge its state estimate before it embarked for the next observable zone. The best sequence of paths determined by the path planner are shown in Figure 5.10 with the resultant path and state estimate shown in Figure 5.11.

These results represent expected outcomes based on simulation. Although the naive navigation performed surprisingly well, the final error was still larger than the path planner error as shown in Figures 5.12 and 5.13 respectively. The performance of the path planner maintained the state estimate position error below 0.22 m, achieving a final error of 0.046 m while the naive navigation error grew to 0.30 m by the end of the trial.

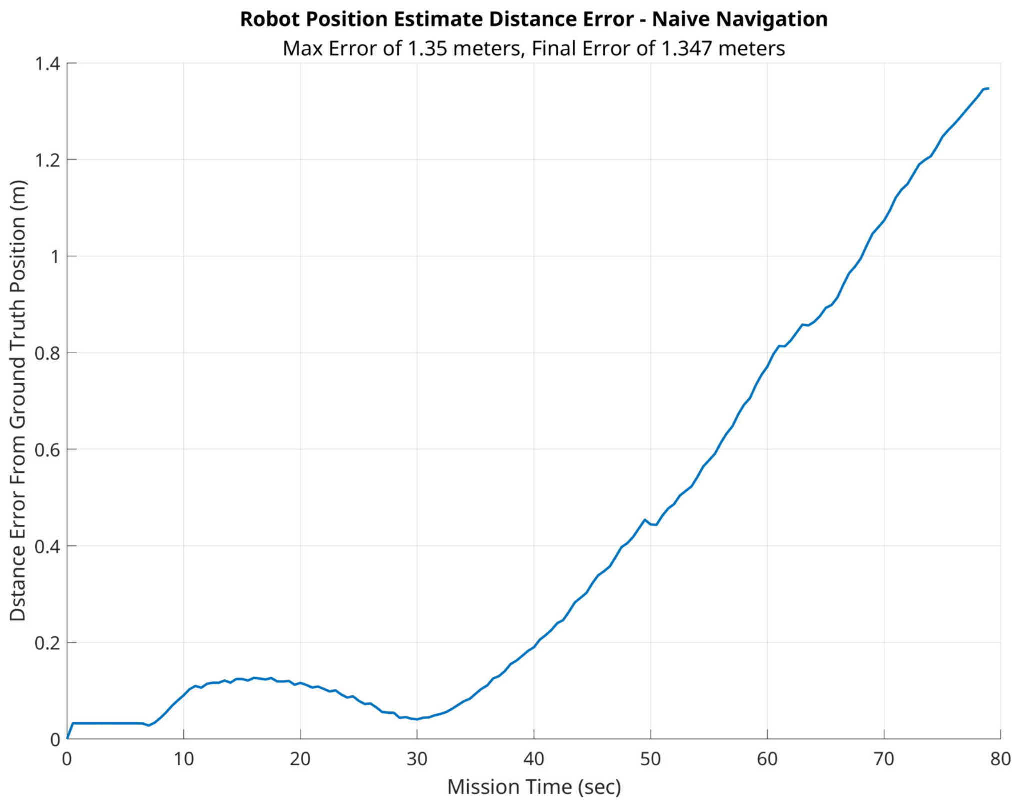

Figure 5.8: Naive Navigator State Estimate Error from Ground Truth

5.3.4 Case Study #3

The third case study demonstrated the limitations of the path planner when the nearest observable zone was beyond the time horizon of the path planner exploration. In this study, the goal was also located at (5,5) and two observable zones were positioned to either side of the direct path toward the goal, each with a radius of 0.5 m and located at (3,5) and (5,3).

In the case of the naive controller heading directly for the the goal, the state estimate steadily diverged as expected, but in this real trial, the robot accidentally encountered one of the observable areas, causing a rapid reconvergence of the state estimate and a low final position error as shown in Figure 5.14.

Initially, the path planner followed a similar trajectory due the lack of observable zones in the vicinity. However, after progressing toward the goal, the

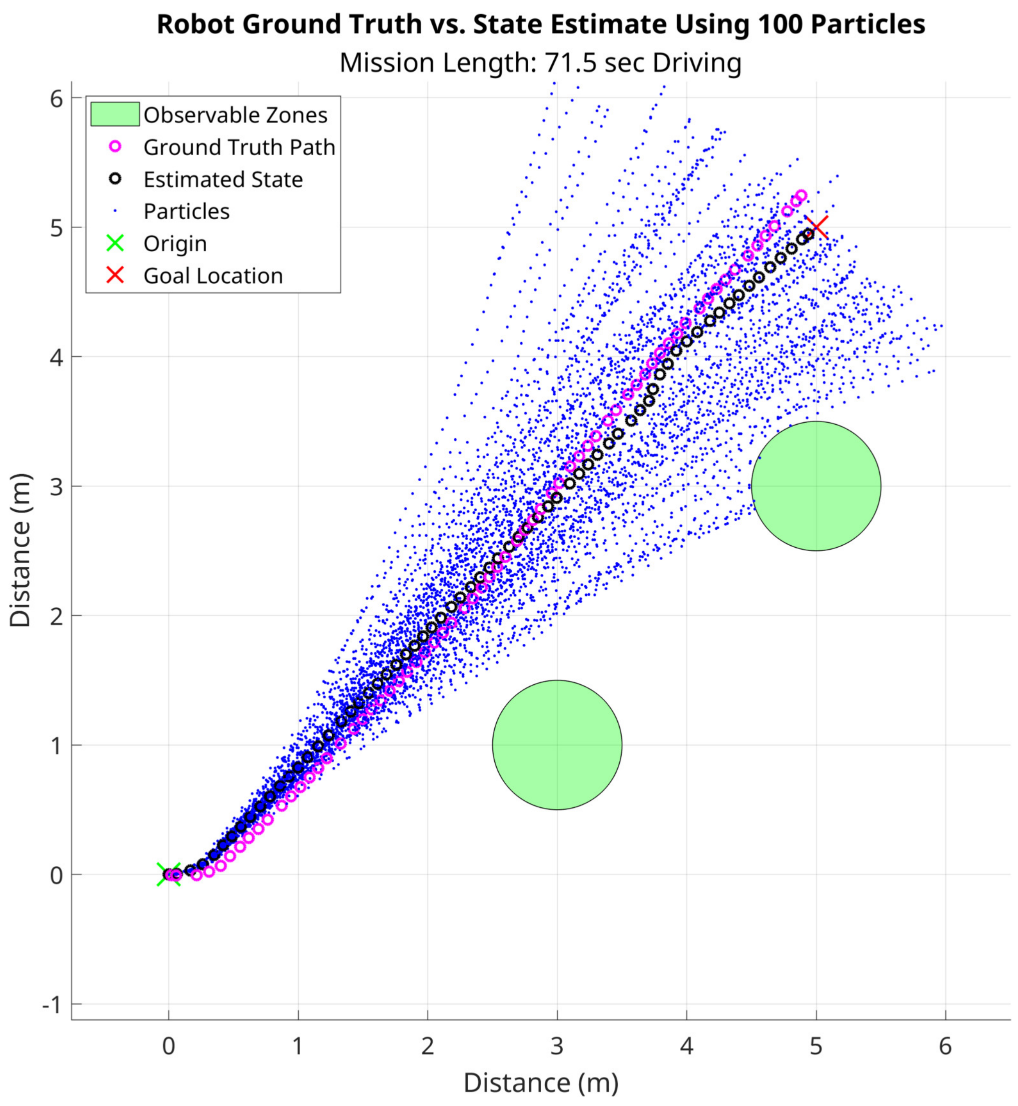

Figure 5.9: Naive Controller State Estimate Navigating Past Observable Zones

observable zones were within range of the path planner time horizon so the robot redirected to seek out the possible of higher certainty state estimates. The best sequence of paths determined by the path planner are shown in Figure 5.15 with the resultant path and state estimate shown in Figure 5.16.

Final error for the path planner is shown in Figure 5.17 while the error for the naive navigation is shown in Figure 5.18 respectively. The performance of the path planner reached a maximum state estimate position error of 0.56 m, achieving a

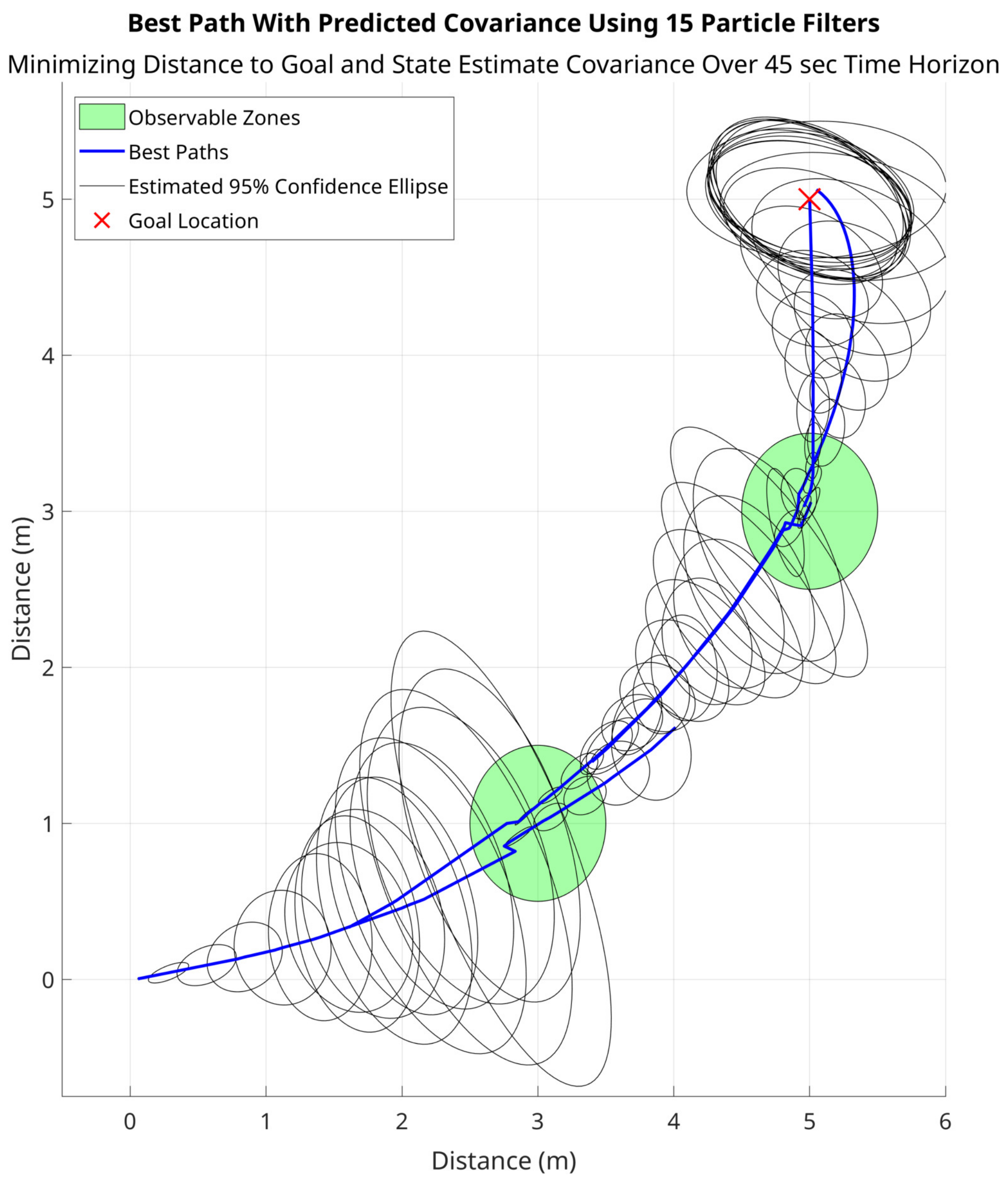

Figure 5.10: Path Planner Selected Paths Projected Through Observable Zones

final error of 0.11 m while the naive navigation error reached a maximum of 0.98 m, achieving a final error of 0.08 m by the end of the trial.

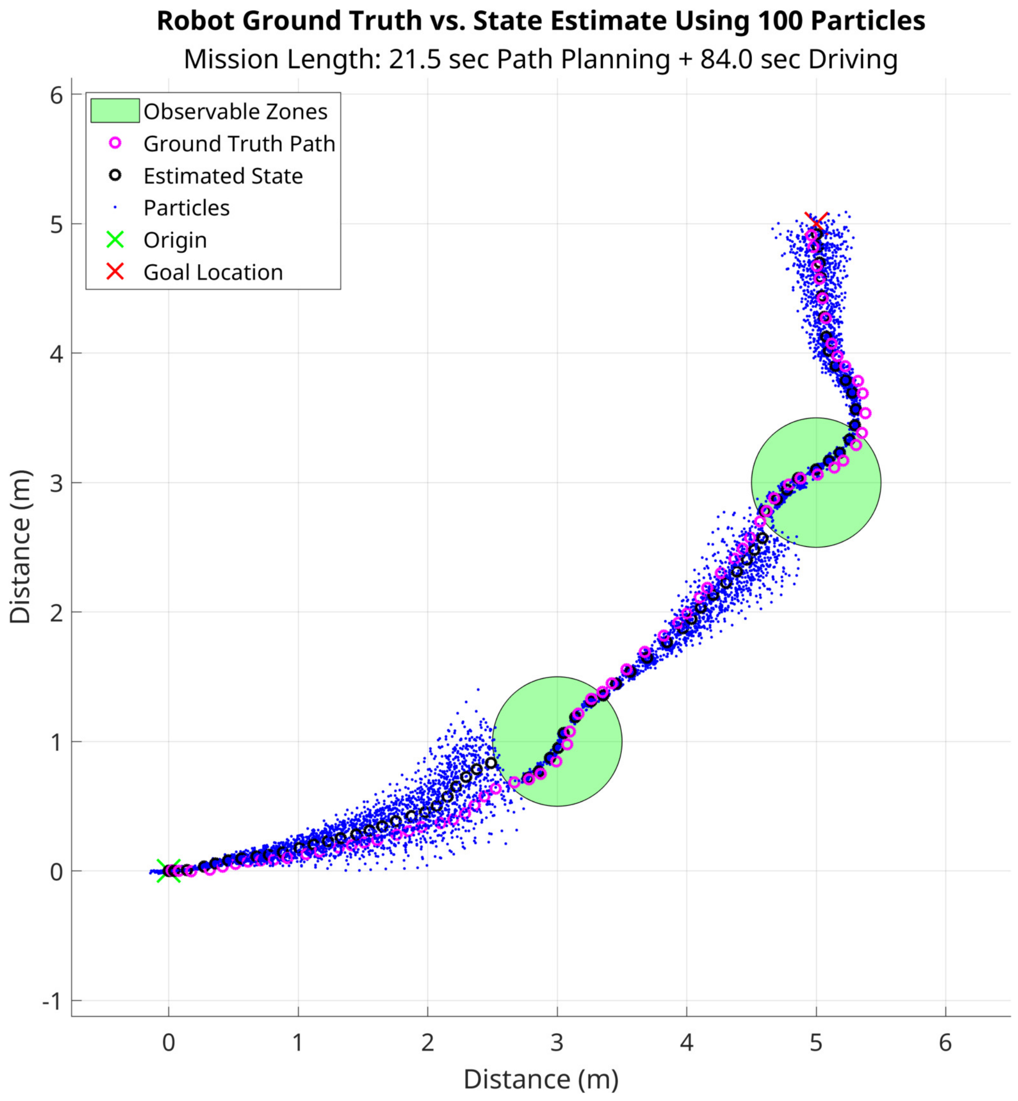

Figure 5.11: Path Planner Robot Navigating Through Observable Zones

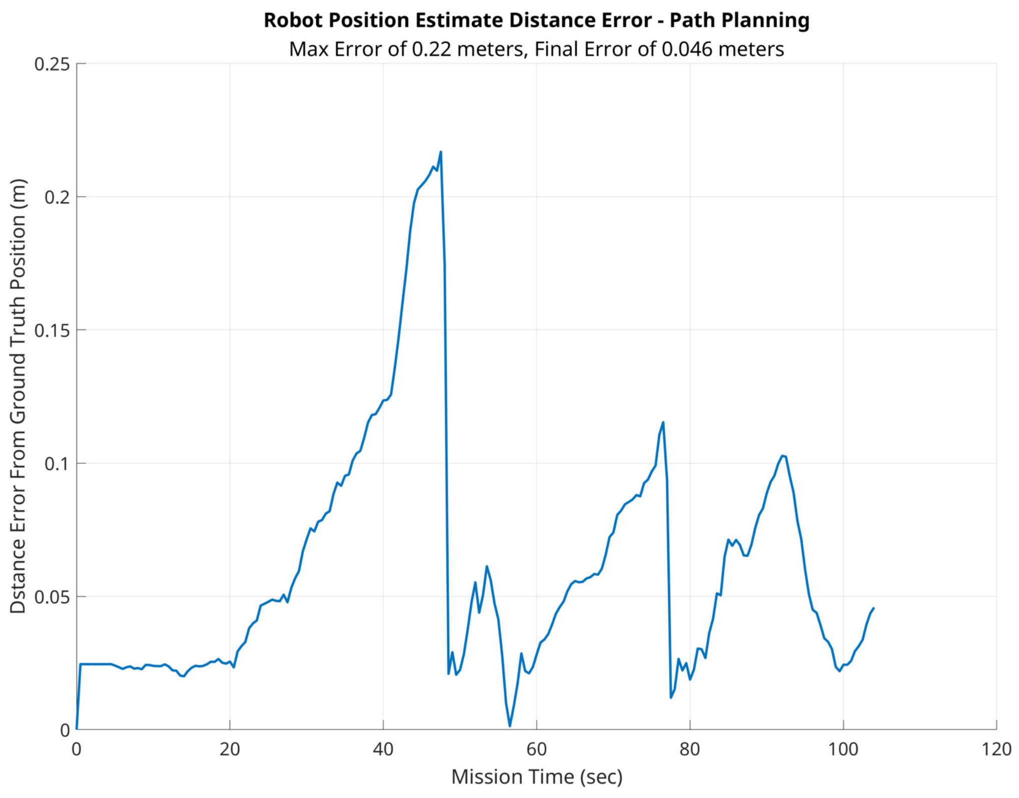

Figure 5.12: Path Planner State Estimate Error from Ground Truth

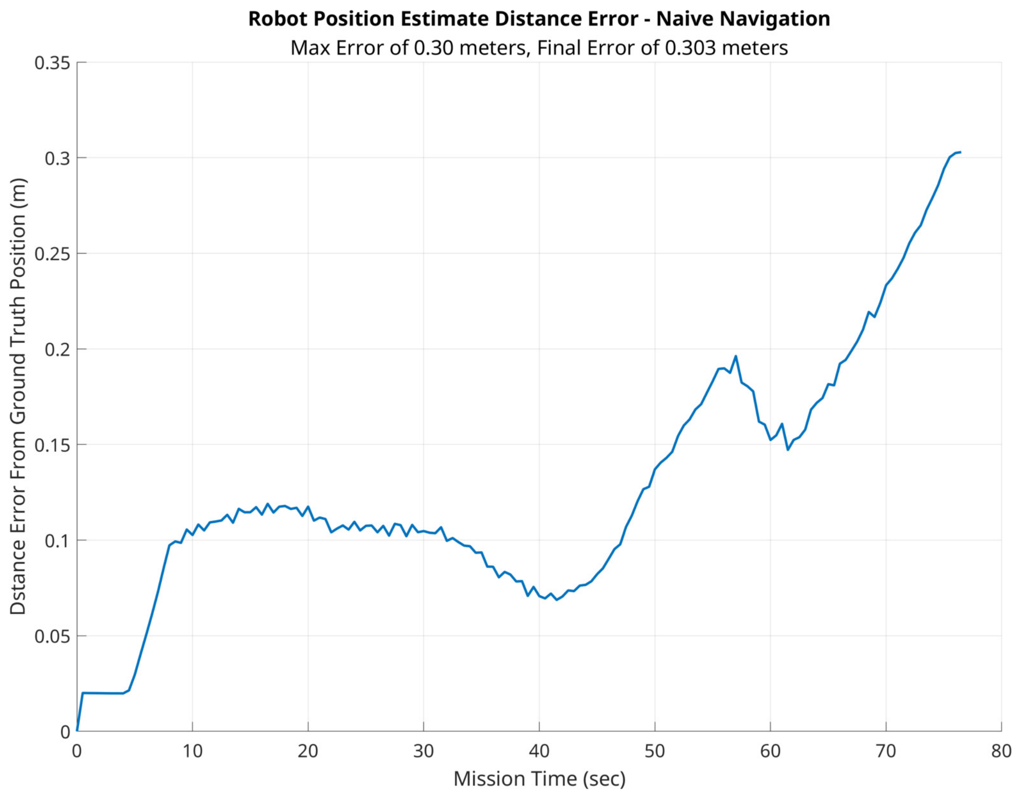

Figure 5.13: Naive Navigator State Estimate Error from Ground Truth

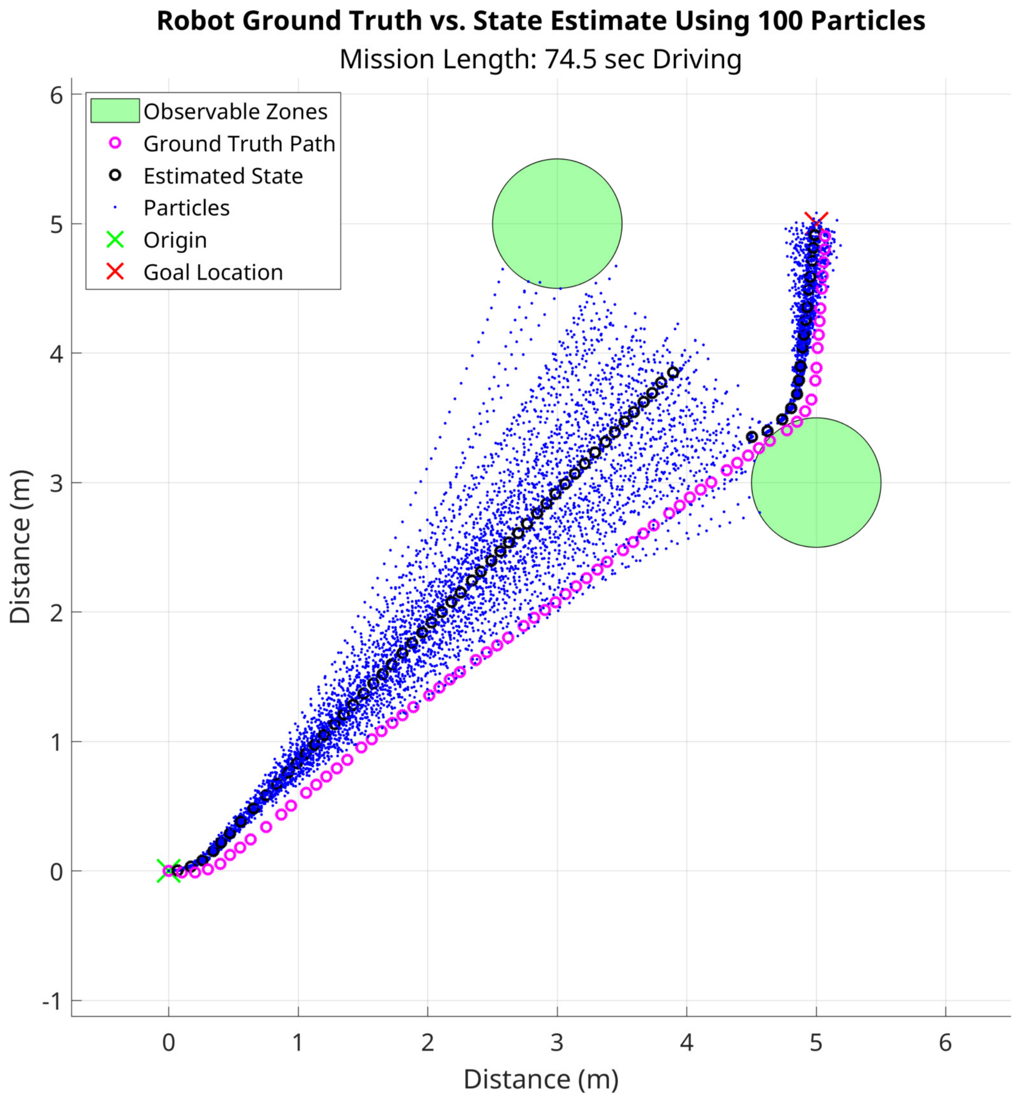

Figure 5.14: Naive Controller State Estimate Navigating Between Observable Zones

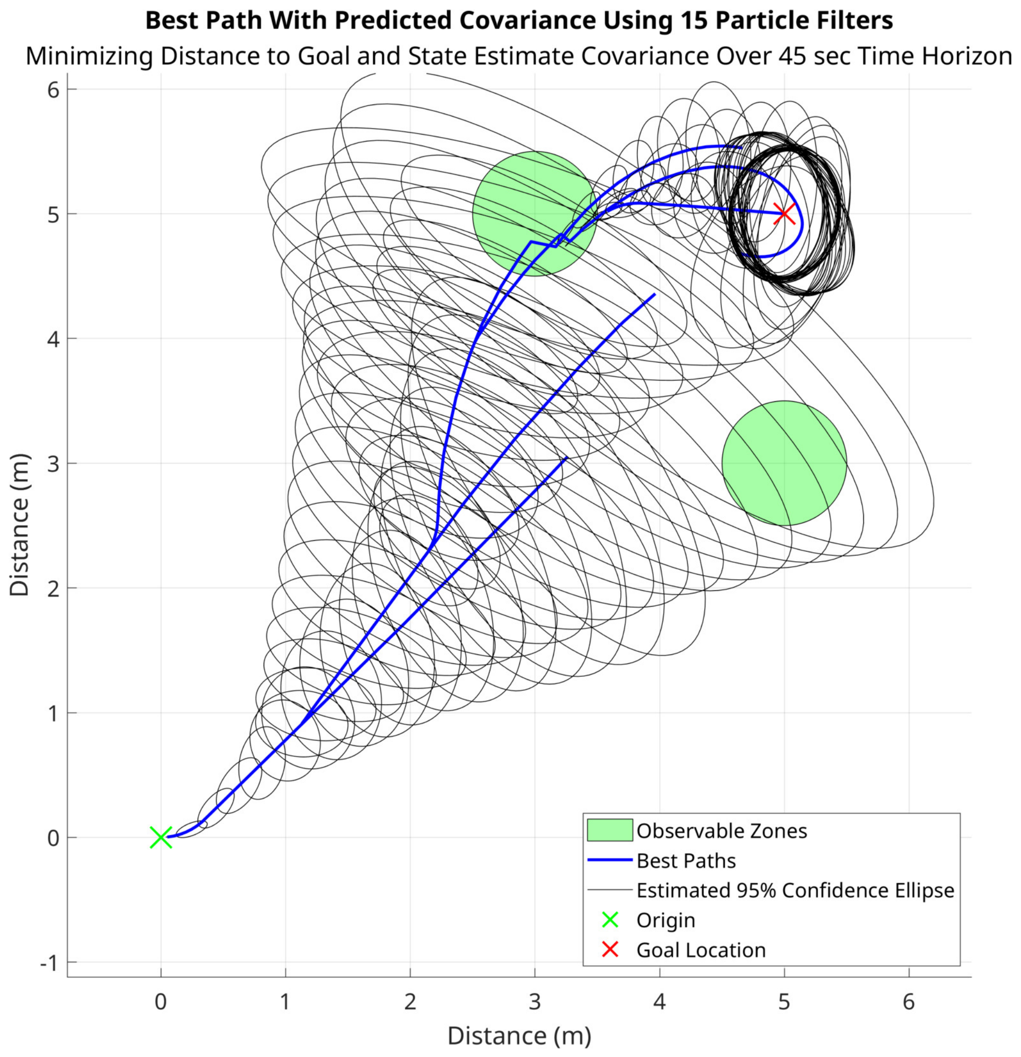

Figure 5.15: Path Planner Selected Paths Projected Through Observable Zones

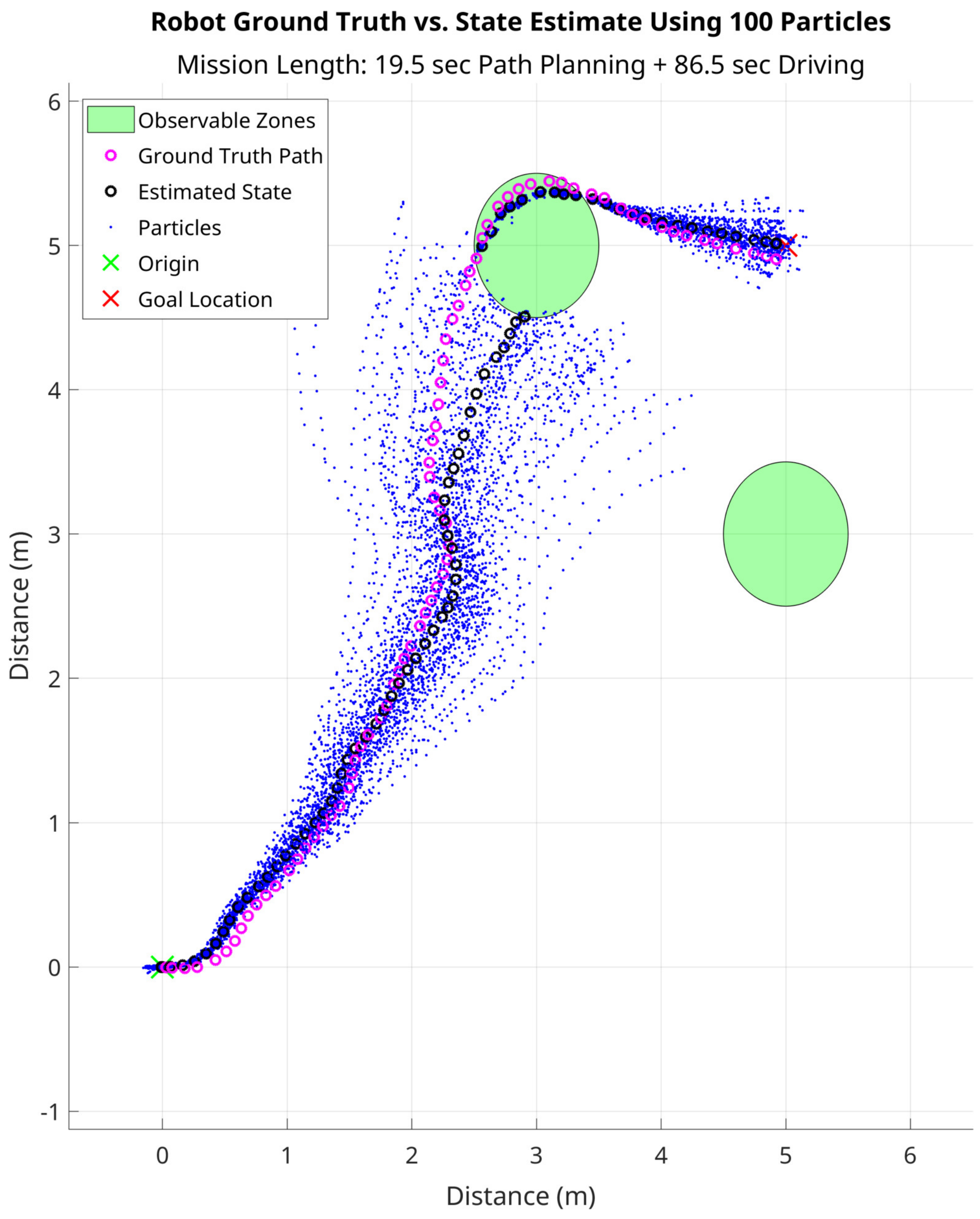

Figure 5.16: Path Planner Robot Navigating Through Observable Zones

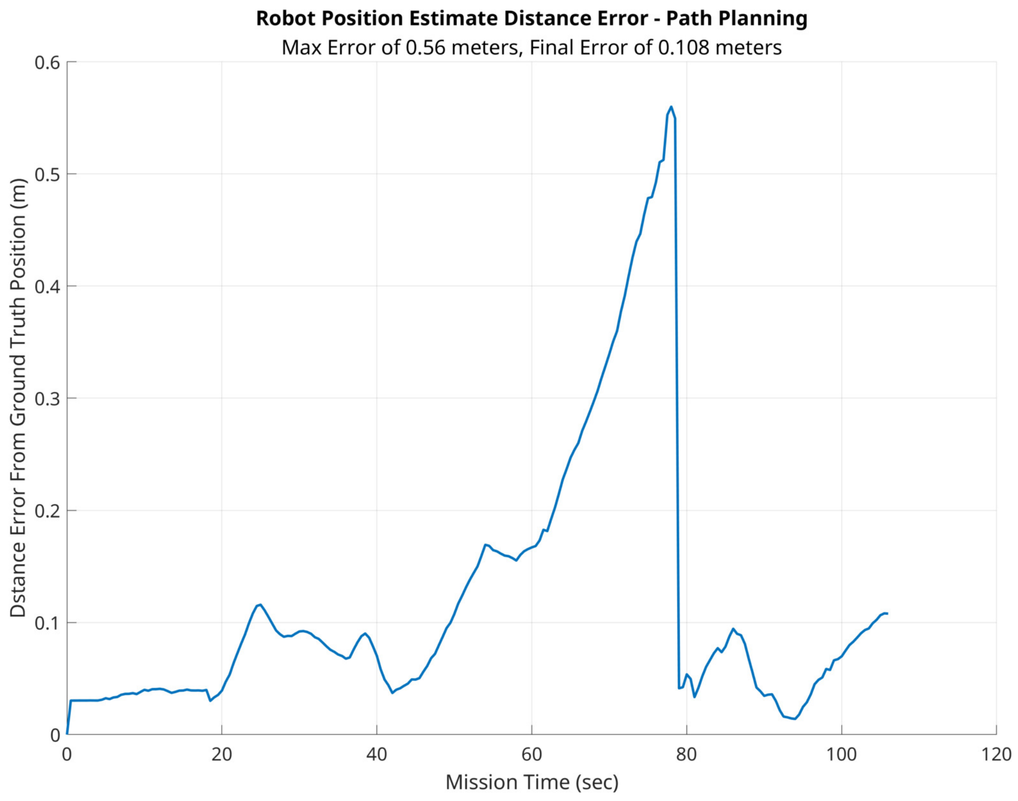

Figure 5.17: Path Planner State Estimate Error from Ground Truth

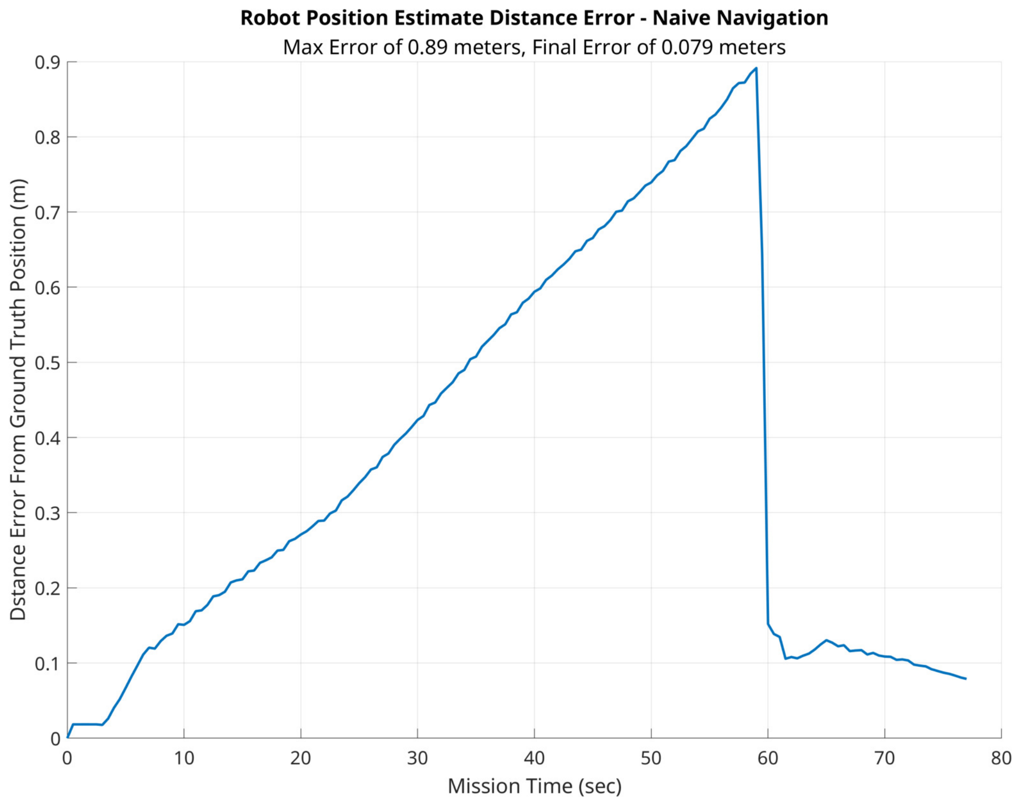

Figure 5.18: Naive Navigator State Estimate Error from Ground Truth

CHAPTER 6

Conclusions

6.1 Results

This research demonstrated that applying model-predictive control to explore and weight potential trajectories in a two dimensional navigation problem in a sparsely observable environment yielded performance improvements over a naive controller that did not consider observability properties. As desired, the path planner was able to find and select paths that traveled to the goal while also minimizing state estimate uncertainty along the way. The state estimates of the path planner robot were not observed to be worse than the naive navigator that directly sought out the goal configuration without considering the benefit of utilizing observable regions. However, this work demonstrated that the path planner took longer to plan and complete the path to the goal due to the priority of state estimation accuracy over rapid progress toward the goal.

6.2 Limitations

Depending on the environment, the path planner's decision to route to the robot to investigate an area with favorable observability or measurement characteristics may give the system the best possible chance at reaching that favorable region compared with a naive approach. However, there may be situations where the nearest observability region is outside of the neighborhood explored by the time horizon, or where the system cannot be reached reliably due to divergence of state certainty. In these cases, the path planner may not perform any better than a standard controller and in fact may perform worse due to the extra time spent searching for an observable region.

Initially, random commands were used to generate paths, but too many paths were required to achieve a reasonable outcome so it was not feasible to run the path planner in real time. Using breadth-first search was a necessary deviation from the original intent of the particle filter application to minimize the number of evaluated paths. The intermediate solution of adding random rotation commands to the breadth-first search was attempted, but also required many times more paths to achieve smooth paths for the robot to follow, again increasing computational complexity beyond the capability for real time application on the hardware used for this research.

6.3 Future Work

This research developed a method for controlling a system in a state space with state dependent observability. This method of projecting and weighting path performance could potentially be applied to situations where measurement quality is on a spectrum and some areas of the state space yield poor measurements rather than the binary case of good or bad measurements evaluated in this work. Additionally, other model-predictive control path weighting factors could be investigated to yield more robust optimization, including the direct use of particle distributions to preserve nonlinear state estimation benefits instead of the Gaussian representation used in this research.

BIBLIOGRAPHY

Askari, I., Zeng, S., and Fang, H., "Nonlinear model predictive control based on constraint-aware particle filtering/smoothing," in *2021 American Control Conference (ACC)*, May 2021, pp. 3532–3537.

Barbehenn, M. and Hutchinson, S., "Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 2, pp. 198–214, April 1995.

Bergman, K., Ljungqvist, O., Glad, T., and Axehill, D., "An optimization-based receding horizon trajectory planning algorithm," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 550–15 557, 2020, 21st IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320330810

Botchu, S. K. and Ungarala, S., "Nonlinear model predictive control based on sequential monte carlo state estimation," *IFAC Proceedings Volumes*, vol. 40, no. 5, pp. 29–34, 2007, 8th IFAC Symposium on Dynamics and Control of Process Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015317663

Bry, A. and Roy, N., "Rapidly-exploring random belief trees for motion planning under uncertainty," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 723–730.

de Villiers, J., Godsill, S., and Singh, S., "Particle predictive control," *Journal of Statistical Planning and Inference*, vol. 141, no. 5, pp. 1753–1763, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378375810005252

Dellaert, F., Fox, D., Burgard, W., and Thrun, S., "Monte carlo localization for mobile robots," in *Proceedings of (ICRA) International Conference on Robotics and Automation*, vol. 2, May 1999, pp. 1322 – 1328.

Doucet, A. and Johansen, A., "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of Nonlinear Filtering*, vol. 12, 01 2009.

Ferguson, D., Likhachev, M., and Stentz, A., "A guide to heuristic-based path planning," 01 2005.

Frey, K. M., Steiner, T. J., and How, J. P., "Towards online observability-aware trajectory optimization for landmark-based estimators," 2020.

Gonzalez Bautista, D., Pérez, J., Milanes, V., and Nashashibi, F., “A review of motion planning techniques for automated vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, pp. 1–11, 11 2015.

Gordon, N. J., Salmond, D., and Smith, A. F. M., “Novel approach to nonlinear/non-gaussian bayesian state estimation,” 1993.

Grebe, C., Wise, E., and Kelly, J., “Observability-aware trajectory optimization: Theory, viability, and state of the art,” in *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, Sep. 2021, pp. 1–8.

Hermann, R. and Krener, A., “Nonlinear controllability and observability,” *IEEE Transactions on Automatic Control*, vol. 22, no. 5, pp. 728–740, 1977.

Kalman, R. E., “A New Approach to Linear Filtering and Prediction Problems,” *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 03 1960. [Online]. Available: https://doi.org/10.1115/1.3662552

Kalman, R., “On the general theory of control systems,” *IFAC Proceedings Volumes*, vol. 1, no. 1, pp. 491–502, 1960, 1st International IFAC Congress on Automatic and Remote Control, Moscow, USSR, 1960. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017700948

Kelly, A. and Nagy, B., “Reactive nonholonomic trajectory generation via parametric optimal control,” *The International Journal of Robotics Research*, vol. 22, no. 7-8, pp. 583–601, 2003. [Online]. Available: https://doi.org/10.1177/02783649030227008

Kinsey, J., Eustice, R., and Whitcomb, L., “A survey of underwater vehicle navigation: Recent advances and new challenges,” 01 2006.

Kitagawa, G., “Non-gaussian state-space modeling of nonstationary time series,” *Journal of the American Statistical Association*, vol. 82, no. 400, pp. 1032–1041, 1987. [Online]. Available: http://www.jstor.org/stable/2289375

Kou, S. R., Elliott, D. L., and Tarn, T. J., “Observability of nonlinear systems,” *Information and Control*, vol. 22, no. 1, pp. 89–99, 1973. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019995873905081

Kuptametee, C. and Aunsri, N., “A review of resampling techniques in particle filtering framework,” *Measurement*, vol. 193, p. 110836, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263224122001312

Liu, C., Lee, S., Varnhagen, S., and Tseng, H. E., “Path planning for autonomous vehicles using model predictive control,” in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 174–179.

Marquez, H., *Nonlinear Control Systems: Analysis and Design.* Wiley-Interscience, 11 2002.

Martinelli, A. and Siegwart, R., "Observability analysis for mobile robot localization," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 1471–1476.

Melo, J. and Matos, A., "Survey on advances on terrain based navigation for autonomous underwater vehicles," *Ocean Engineering*, vol. 139, pp. 250–264, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002980181730241X

Prentice, S. and Roy, N., "The belief roadmap: Efficient planning in linear pomdps by factoring the covariance," in *International Symposium of Robotics Research*, 2007.

Raković, S. V. and Levine, W. S., Eds., *Handbook of Model Predictive Control.* Birkhäuser Cham, 2019.

Sang, H., You, Y., Sun, X., Zhou, Y., and Liu, F., "The hybrid path planning algorithm based on improved a* and artificial potential field for unmanned surface vehicle formations," *Ocean Engineering*, vol. 223, p. 108709, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002980182100144X

Snyder, C., "Particle filters, the optimal proposal and high-dimensional systems," Ph.D. dissertation, Shinfield Park, Reading, 2012 2012.

Stahl, D. and Hauth, J., "Pf-mpc: Particle filter-model predictive control," *Systems and Control Letters*, vol. 60, no. 8, pp. 632–643, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167691111001125

Thrun, S., Burgard, W., and Fox, D., *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

van den Berg, J., Abbeel, P., and Goldberg, K., "Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 895–913, 2011. [Online]. Available: https://doi.org/10.1177/0278364911406562

Wilson, A. D., Schultz, J. A., and Murphey, T. D., "Trajectory synthesis for fisher information maximization," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1358–1370, Dec 2014.

Zhou, C., Huang, B., and Fränti, P., "A review of motion planning algorithms for intelligent robotics," 2021.