Combining Local and Global Optimization for Planning and Control in Information Space

by

Vu Anh Huynh

Submitted to the School of Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Computation for Design and Optimization

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2008

Author .....
School of Engineering
August 14, 2008

Certified by .....
Nicholas Roy
Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Jaime Peraire
Professor of Aeronautics and Astronautics
Codirector, Computation for Design and Optimization Program

Combining Local and Global Optimization for Planning and Control in Information Space

by

Vu Anh Huynh

Submitted to the School of Engineering
on August 14, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science in Computation for Design and Optimization

Abstract

This thesis presents a novel algorithm, called the parametric optimized belief roadmap (POBRM), to address the problem of planning a trajectory for controlling a robot with imperfect state information under uncertainty. This question is formulated abstractly as a partially observable stochastic shortest path (POSSP) problem. We assume that the feature-based map of a region is available to assist the robot's decision-making.

The POBRM is a two-phase algorithm that combines local and global optimization. In an offline phase, we construct a belief graph by probabilistically sampling points around the features that potentially provide the robot with valuable information. Each edge of the belief graph stores two transfer functions to predict the cost and the conditional covariance matrix of a final state estimate if the robot follows this edge given an initial mean and covariance. In an online phase, a sub-optimal trajectory is found by the global Dijkstra's search algorithm, which ensures the balance between exploration and exploitation. Moreover, we use the iterative linear quadratic Gaussian algorithm (iLQG) to find a locally-feedback control policy in continuous state and control spaces to traverse the sub-optimal trajectory.

We show that, under some suitable technical assumptions, the error bound of a sub-optimal cost compared to the globally optimal cost can be obtained. The POBRM algorithm is not only robust to imperfect state information but also scalable to find a trajectory quickly in high-dimensional systems and environments. In addition, the POBRM algorithm is capable of answering multiple queries efficiently. We also demonstrate performance results by 2D simulation of a planar car and 3D simulation of an autonomous helicopter.

Thesis Supervisor: Nicholas Roy
Title: Assistant Professor of Aeronautics and Astronautics

Acknowledgments

This thesis could not have been completed without the support of numerous individuals. I would like to thank my thesis advisor, Nicholas Roy, for his guidance and support. I gained a lot from every discussion we had.

My sincere gratitude goes to the CDO faculties and the fellow students in my class and research group. It is especially enjoyable to explore several research topics and have delicious food in weekly group meetings in CSAIL. I would also like to thank all the new friends I have made in MIT, who have made my life in MIT so wonderful.

I also gratefully acknowledge the generous financial support from the Singapore MIT Alliance.

Finally, I would like to thank my family for their unconditional love and encouragement.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

1.1 Motivations

Sequential decision-making is one of the central problems in artificial intelligence and control theory fields [19]. In domains where perfect knowledge of the state of an agent is available, a Markov decision process (MDP) is effective for modeling the interaction between an agent and a stochastic environment. Many algorithms have been studied extensively to find an optimal policy for a small-size to middle-size MDP [3] [29]. These algorithms often model state and control spaces as discrete spaces. However, in most real world problems such as robot navigation, the state of the agent is not fully observable at all times, in which case a partially observable Markov decision process (POMDP) is used to model interactions of the agent. In a POMDP, the system dynamics are known *a priori*, similar to an MDP, but the state of the agent must be inferred from the history of observations. The agent can maintain a distribution, called the agent's belief distribution, over states to summarize the entire history [19].

The problem of planning and controlling a robot with limited sensors is a challenging question. In particular, flying unmanned aerial vehicles (UAVs) in GPS-denied environments such as indoor buildings or urban canyons has received increasing attention from research communities, industry and the military. With limited on-board sensors such as sonar ranging, laser ranging, and video imaging, UAVs acquire observations to estimate their positions for decision-making and control. Moreover, the

states and control signals of UAVs are naturally in continuous spaces. Therefore, solving the problem of UAV navigation in this case inevitably leads to a complex continuous POMDP [30] [8].

Despite extensive research in recent years, finding an optimal policy in a POMDP is still a challenge for three main reasons. First, although we can convert a POMDP in state space to an MDP in belief space, available MDP algorithms for discretized belief space are computationally expensive even for small problems because the number of discretized belief states grows rapidly when agents interact with environments over time [19].

Second, an agent's policy must use available information to achieve the task at hand, which is known as exploitation, and, at the same time, take actions that further acquire more confidence in its state, which is known as exploration. Approximate approaches, such as certainty equivalence (CE), that use MDP algorithms to work on the means of state estimates instead of the entire belief space do not address well the balance between exploitation and exploration [33] [1].

Third, other attempts to use gradient-like algorithms with parameterized state and control spaces are often subject to local minima [19] [3] [29]. Most of current methods do not directly evaluate the quality of a sub-optimal solution, or the provided error bound is not tight enough [21]. Therefore, the error bound of a sub-optimal objective value compared to the globally optimal objective value still remains an open question.

1.2 Problem statement and objectives

In this research, we consider the problem of planning and controlling a robot to navigate from a starting location to a specific destination in a partially observable stochastic world. Abstractly, this problem can be considered as a partially observable stochastic shortest path (POSSP) problem [3] [4] [34]. The robot operates in environments where global localization information is unavailable. Instead, the robot is equipped with on-board sensors having limited range or field of view to estimate its

positions. Moreover, the robot is able to access to the feature-based map of an operating environment. Each feature in the map is fully detected with known coordinates. In addition, the stochastic dynamics of the robot and sensors are given. Under these assumptions, our approach, the parametric optimized belief roadmap (POBRM), addresses the above three difficulties to plan and control the robot directly in continuous state and control spaces.

1.3 Approach and contributions

1.3.1 Approach

The POBRM is a two-phase algorithm, with an offline phase and an online phase. In the offline phase, we construct a belief graph by probabilistically sampling points around the features that potentially provide the robot with valuable information. Each edge of the belief graph stores two transfer functions to predict the cost and the conditional covariance matrix of a final state estimate if the robot follows this edge given an initial mean and covariance. The transfer function to predict the cost to traverse an edge is found by iterative stochastic algorithms, which are based on the least square curve-fitting method [4]. The transfer function to predict the conditional covariance of a final state estimate at an incoming vertex is based on the property that the covariance of the extended Kalman filter (EKF) can be factored, leading to a linear update in the belief representation [22] [8].

In the online phase, an approximation of an optimal trajectory is found by searching the belief graph using the global Dijkstra's search algorithm. We then refine the approximated trajectory using the iterative linear quadratic Gaussian algorithm (iLQG) and find a locally-feedback control policy in continuous state and control spaces [15]. Thus, the POBRM algorithm combines both local and global optimization in the offline and online phases to plan and control in belief space.

1.3.2 Contributions

The contributions of this thesis are three-fold. First, the POBRM algorithm aims at moving the computational burden of an induced optimization problem into the offline phase. The offline phase is computed only once for a particular map, and therefore it will save time in every online Dijkstra's search phase. In addition, the Dijkstra's search ensures the balance between exploration and exploitation. Compared to other approaches, the POBRM algorithm is not only robust to imperfect state information but also highly scalable to apply in high-dimensional systems and large-scale environments. Moreover, the POBRM algorithm is capable of answering multiple queries efficiently.

Second, we prove that if the belief graph contains edges along a globally optimal trajectory, by using the iLQG in the online phase, the error bound of a sub-optimal cost compared to the globally optimal cost can be obtained. Thus, in principle, the POBRM algorithm is able to provide a mechanism to evaluate the quality of a sub-optimal solution.

Third, to the best of our knowledge, it is the first time in this thesis that the problem of planning and controlling a six-degree-of-freedom helicopter with *coastal navigation* trajectories [25] in continuous spaces is reported.

1.4 Thesis outline

The chapters of the thesis are organized as follows:

- Chapter 2 presents the overview of related work in artificial intelligence and control theory fields. This chapter introduces the fundamental of Markov decision processes in both fully observable and partially observable cases to solve this class of problems. In addition, the emphasis of exploitation and exploration in the partially observable case is presented. Also, related methods in dynamic programming (DP) such as iterative linear quadratic Gaussian (iLQG) are discussed. Moreover, advances in mobile robotics, especially UAVs, are introduced.

This chapter also provides the mathematical formulation of the problem under consideration. An abstract framework with continuous-time and discrete-time formulations is presented first. Then, concrete system dynamics and sensor dynamics of a two-dimensional (2D) planar car and a three-dimensional (3D) helicopter follow.

- Chapter 3 discusses the POBRM algorithm. Most importantly, techniques in the offline phase and the online phase are developed in detail. In this chapter, we keep the algorithm in the most generic framework when presenting the iLQG algorithm to design a sub-optimal controller. When we start constructing a belief graph for a given feature-based map, we will focus more on the context of planning and controlling mobile robots. We then provide the preliminary theoretical analysis of the POBRM algorithm. We show that under suitable assumptions, the cost of a globally optimal trajectory can be bounded. The analysis in this chapter can serve as a theoretical basis for future work.

- Chapter 4 presents the results of experiments in simulation. In particular, 2D experiments simulate a mobile robot in a planar world, and the 3D experiments simulate a six-degree-of-freedom helicopter. Several results are shown to verify the theoretical results in Chapter 3.

- Chapter 5 discusses the main conclusions of this research. Also, several future research directions are suggested.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Literature Review and Problem Formulation

In this chapter, several important research concepts that are relevant to this work are presented. This review also sets the context for this work. To formulate the problem, we first state a generic continuous-time formulation, and then present the discrete-time approximation of the generic system. We next state additional assumptions to make the problem more tractable in this research, and give an induced optimization problem. Finally, we present the two-dimensional (2D) model of a planar car and the three-dimensional (3D) model of a helicopter.

2.1 Sequential decision making

Different research communities often address similar problems that arise independently in their fields. This is also the case for the problem of sequential decision making in artificial intelligence (AI) and control theory communities. In AI, researchers want to design an agent that can interact with an outside environment to maximize a long-term reward function. Traditionally, AI researchers often deal with discrete spaces in problems such as playing chess. Similarly, in control theory, researchers want to autonomously control a system in a certain environment to optimize some criteria. Researchers in this field often deal with continuous problems such as controlling a

mobile robot or guiding a missile [19].

Nevertheless, the above problems in AI and control theory share much in common. There are corresponding terminologies between AI and control theory. In particular, the terms *agent, environment*, and *maximum reward* in AI correspond respectively to *controller, plant*, and *minimum cost* in control theory [19]. In fact, recent research trends use techniques in both fields to address hard problems such as path planning and control for a mobile robot. One of these techniques is based on dynamic programming from which a large variety of algorithms have been developed to address different characteristics of different problems. We will overview these techniques subsequently.

2.1.1 Markov decision process

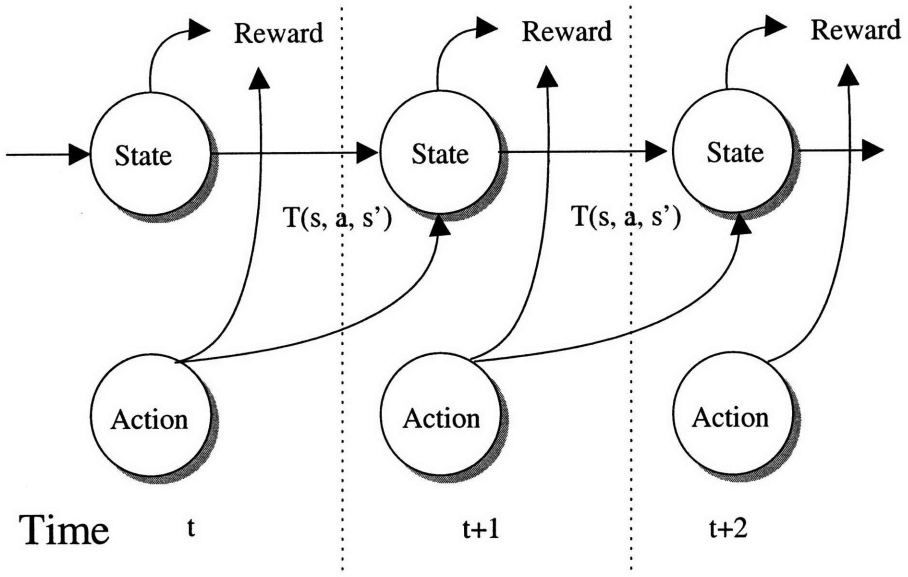


Figure 2-1: Graphical representation of an MDP (image courtesy of Nicholas Roy).

As mentioned in Chapter 1, a Markov decision process (MDP) is a common model of sequential decision making problems in both AI and control theory. An MDP is a model that specifies how an agent interacts with a stochastic environment with a known transition model. The components of an MDP are [24]:

- A set of states S,
- A set of actions A,

- A set of transition probabilities $T : S \times A \times S \mapsto [0, 1]$ s.t. $T(s, a, s') = P(s'|s, a)$,
- A set of reward functions $R : S \times A \mapsto R$,
- A discount factor γ,
- An initial state s_0.

Figure 2-1 depicts graphically how these components relate with each other in an MDP. As can be seen here, at each state, the agent chooses an action, which results in a reward value and causes the state to change. The posterior state obeys the stochastic transition probability T. Furthermore, this figure also illustrates that the system is a first-order Markov graph. In other words, the distribution of a future state value depends only on the current state value and control input but not past state values and control inputs (Definition 2.1.1, 2.1.2).

Definition 2.1.1. *(Markov property) If x(t) and u(t) are two stochastic processes, the Markov property states that, with $x, \chi \in S$, and $u \in A$,*

$$\begin{aligned} &P\left[x(t+h) = \chi(t+h) \middle| x(s) = \chi(s), u(s) = v(s), \forall s \leq t\right] \\ &= P\left[x(t+h) = \chi(t+h) \middle| x(t) = \chi(t), u(t) = v(t)\right], \quad \forall h \geq 0. \end{aligned}$$

Definition 2.1.2. *(Discrete-time Markov property) If x_k and u_k are two discrete stochastic processes, the Markov property states that, with $x, \chi \in S$, and $u \in A$,*

$$\begin{aligned} &P\left[x_{t+h} = \chi_{t+h} \middle| x_s = \chi_s, u_s = v_s, \forall s \leq t\right] \\ &= P\left[x_{t+h} = \chi_{t+h} \middle| x_t = \chi_t, u_t = v_t\right], \quad \forall h \geq 0. \end{aligned}$$

Methods to solve an MDP are investigated extensively in [3] [29] [4]. Notable methods are value iteration and policy iteration, which are based on Bellman's equation. The value iteration method iteratively finds the optimal objective values associated with possible states. From this value function, a suitable policy is then retrieved. Alternatively, the policy iteration method iteratively finds an optimal control policy

by evaluating the current policy and making policy improvements. Recently, Wang et al. have proposed a dual approach that explicitly maintains a representation of stationary distributions as opposed to value functions to solve an MDP [36] [35]. This yields novel dual forms of the value iteration and policy iteration.

2.1.2 Partially observable Markov decision process

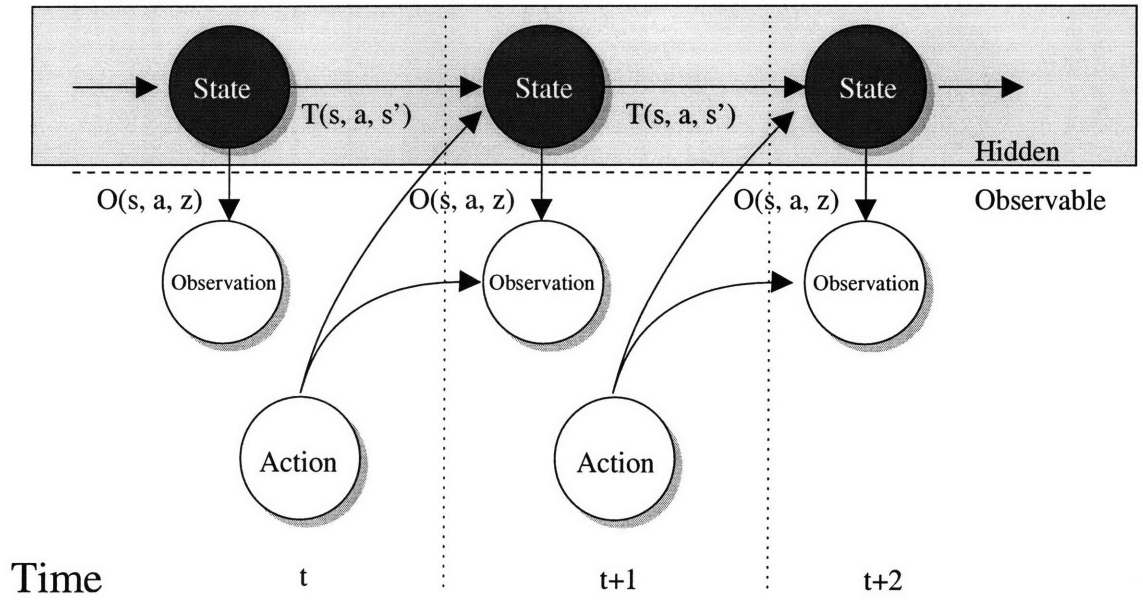


Figure 2-2: Graphical representation of a POMDP (image courtesy of Nicholas Roy).

Although an MDP is suitable for problems whose states are observable at all times such as playing chess, it is insufficient to model problems whose states are not directly observable. In such cases, a POMDP, which has additional observations, is suitable for modeling those problems [19] [24]. Components of a POMDP are:

- A set of states S,
- A set of actions A,
- A set of transition probabilities $T : S \times A \times S \mapsto [0, 1]$ s.t. $T(s, a, s') = P(s'|s, a)$,
- A set of observations Z,

- A set of observation probabilities $O : S \times A \times Z \mapsto [0, 1]$ s.t. $O(s, a, z) = P(z|s, a)$,
- A set of reward functions $R : S \times A \times Z \mapsto R$,
- A discount factor γ,
- An initial state distribution $P(s_0)$.

Compared to an MDP, there are an addition set of observations Z and corresponding probabilities. Again, these components can be represented graphically in Figure 2-2. As can be seen, black states are inside the shaded box to denote that they cannot be directly accessed but must be inferred from the history of observations. Therefore, these states are called hidden states.

Methods that solve a POMDP with discrete states, discrete actions, and discrete observations are discussed in detail in [19] [30] [24]. The simplest approach is also based on Bellman's equation [30], which computes a sequence of value functions, for increasing planning horizons. At each horizon, the value function is represented as a piece-wise linear function, where the number of linear pieces increases rapidly with the horizon length. Point-based Value Iteration (PBVI) [21] [28] can be applied to prune unnecessary components. Using the PBVI method, Pineau et al. provide the error bound over multiple value updates, which depends on how dense the samples in the belief space are [21]. Kurniawati et al. have recently proposed a new point-based algorithm SARSOP, which stands for successive approximations of the reachable space under optimal policies [11]. This algorithm exploits the notion of optimally reachable belief space to compute efficiently.

To avoid the intractability of the exact POMDP value iteration, the belief space can be gridded using a fixed grid [17] [5] or a variable grid [6] [38]. Value backups are computed at grid points, and the gradient is ignored. The value of non-grid points are inferred using an interpolation rule.

There is a popular heuristic technique Q_{MDP} [16] that treats a POMDP as if it were fully observable and solves a related MDP. The Q_{MDP} algorithm can be effective

in some domains, but it will fail in domains where repeated information gathering is necessary. Heuristic search value iteration (HSVI) [27] is an approximate value iteration method that performs local updates based on upper and lower bounds of the optimal value function. Alternatively, forward search value iteration (FSVI) [26] is a heuristic method that utilizes the underlying MDP such that actions are chosen based on the optimal policy for the MDP, and then are applied to the belief space.

Other approaches employ approximation structures, including neural networks, to compute the value function or control policy for a subset of states and infer corresponding values for the rest of states [19]. Overall, these methods are computationally expensive.

Approaching from the local solution point of view, Li and Todorov have recently proposed the iLQG algorithm [13] [14] [15] to address the problem of control in partially observable environments, which can be regarded as a POMDP. The main idea of the iLQG algorithm is to find a sub-optimal control policy around the vicinity of some nominal trajectory instead of an optimal control policy for the entire state space. The nominal trajectory is iteratively refined to approach an optimal trajectory under some technical conditions. This method can provide a reasonable solution within an acceptable running time, which makes it attractive in real-time applications.

2.1.3 Exploration versus exploitation

The reason a POMDP is harder to solve than is an MDP is clearly because of the inaccessibility of the true state of a system. While an action in an MDP is chosen to maximize some reward function directly, an action in a POMDP balances two purposes: maximizing the reward function and gaining more confidence in its states. These two purposes were mentioned as exploitation and exploration in Chapter 1.

Tse et al. first mentioned the trade-off between exploitation and exploration in their series of papers in the 1970s [33] [1]. They called this the dual effects of caution (exploitation) and probing (exploration). On one hand, at any moment, if a controller focuses only on the current task, it will prefer a greedy way to achieve the task, but there will be a large uncertainty about accomplishing the task. On

the other hand, at any moment, if the controller focuses totally on gaining more confidence about its current state, the controller will seek actions providing more state information without accomplishing the current task. The optimal controller should balance between exploitation and exploration with the appropriate definition of an objective function. This balance has been examined by many others [29] [18], but as we will show in Chapter 3, in this research, we aim at balancing these two factors during the online phase of the POBRM algorithm with the assistance of the offline phase.

2.2 Advances in mobile robotics

The problem of making an autonomous mobile robot has been studied extensively by both AI and control theory control communities. Several fundamental methods, with many unique characteristics in the robotics field, are presented in [30]. Traditional robot planning methods like the Probabilistic Roadmap (PRM) algorithm [10] assumes full accessibility of all parameters and dynamics of the robot and the environment. Since robots are becoming more and more autonomous but have limited sensors, such powerful accessibility is not always available.

In fact, controlling mobile robots in partially observable stochastic environments is still an on-going research topic. When maps are not available, the fundamental task is to determine the location of environmental features with a roving robot. This is called the simultaneous mapping and localization (SLAM) problem [30] [31]. With a given map, performing autonomous navigation is still challenging. Recently, Roy et al. [22] [8] [25] proposed the *coastal navigation* trajectories to make use of an available map to plan a set of waypoints to a destination. In their work, the main idea is inspired by the ship navigation in the ocean. That is, a ship should navigate along the coast to gain more confidence instead of taking the shortest trajectory with large uncertainty about reaching a destination. These methods can efficiently plan a sub-optimal trajectory even in large-scale environments. However, they only consider minimizing the final uncertainty at a destination but not consumed energy.

Furthermore, they do not provide a corresponding control law to follow this trajectory. In this thesis, we use a similar strategy to plan a sub-optimal trajectory, and at the same time, we provide a sub-optimal control law to follow this trajectory with more complex objective functions. Moreover, there is an important emerging research direction to learn all parameters and dynamics of robots and environments. This class of methods is within the regime of neuro-dynamic programming [4], or reinforcement learning [29]. In this research, although we assume robot dynamics and environments are given, we still apply some of these methodologies to learn part of the cost objective values.

Among many applications of autonomous robots, unmanned aerial vehicles (UAVs) have great potential. In particular, autonomous helicopters or quad-helicopters are useful in missions such as rescuing, monitoring, tracking due to their flexible flight dynamics and simple support infrastructure. Autonomous UAV flight in outdoor environments without GPS signals like forests is illustrated in [12]. It has been demonstrated that a controller can be designed to perform advanced and complicated maneuvers like inverted hovering [20]. The study of the flight dynamics of a quad-helicopter is presented in [9]. In this work, we show how the POBRM algorithm is applied in controlling a six-degree-of-freedom quad-helicopter with simplified flight dynamics in GPS-denied environments, as presented in Chapter 4.

2.3 Generic continuous-time problem

We consider a stationary continuous-time non-linear stochastic dynamic system governed by the following ordinary differential equation (ODE):

$$dx(t) = f(x, u)dt + dw(t), \tag{2.1}$$

where

- $x(t) \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ is the state of the system and summarizes past information that is relevant for future optimization,

- $u(t) \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$ is the control input, or decision variable, to be optimized at any time instant t,
- $w(t) \in \mathbb{R}^{n_w}$ is Brownian noise,
- $f : \mathbb{R}^{n_x+n_u} \mapsto \mathbb{R}^{n_x}$ is a non-linear function of the state and control input.

In the above system, the state $x(t)$ is an element of a space $\mathcal{X}$ that represents state constraints. For instance, in the usual flight of a quad-helicopter, $\mathcal{X}$ is the set of poses with the pitch angle less than 60 degree. Similarly, the control $u(t)$ is an element of a space $\mathcal{U}$ that is a set of admissible controls. It is worthwhile pointing out that states x satisfy the Markov property, which is mathematically stated in Definition 2.1.1.

Moreover, the state of the system is partially observable, and must be inferred from measured observations generated according to

$$y(t) = g(x) + \vartheta(t), \tag{2.2}$$

where

- $y(t) \in \mathbb{R}^{n_y}$ is the measurement output of the system,
- $\vartheta(t) \in \mathbb{R}^{n_\vartheta}$ is Brownian noise,
- $g : \mathbb{R}^{n_x} \mapsto \mathbb{R}^{n_y}$ is a non-linear function of the state.

Eq. 2.2 represents one sensor model. If a robot has many different sensors, there are several models to describe these sensors.

We are interested in finding a feedback control law π^* that minimizes the following objective function, also called the cost-to-go function, from starting time 0 to unspecified final time $\mathcal{T}$ given an initial observation $I(0) = y(0)$ of a starting state $x(0)$:

$$J_\pi\big(I(0)\big) = E\left[h\big(x(\mathcal{T})\big) + \int_0^{\mathcal{T}} \ell\Big(t, x(t), \pi\big(t, I(t)\big)\Big)dt \Big| I(0)\right], \tag{2.3}$$

$$\pi^* = arg\min_{\Pi} J_\pi\big(I(0)\big), \tag{2.4}$$

where

- $h\big(x(\mathcal{T})\big)$ is the final stage cost,
- $\ell\big(t, x, u\big)$ is the instantaneous cost with control $u(t)$ at state $x(t)$,
- $u(t) = \pi\big(t, I(t)\big)$, with $I(t) = \{y(0..t), u(0..t^{-})\}$,
- Π is a set of admissible control laws π.

The conditional expectation is taken over the conditional distribution of $x(0)$ given $I(0)$, and noise processes w, ϑ. The information $I(t)$ stores all measurements up to and including the current measurement as well as all past control inputs. Using this information storage, a feedback control law π decides a control signal at the time instant t through $u(t) = \pi(t, I(t))$. As this formula reads, this is a non-stationary control law that may vary with respect to time.

We can consider this problem as a partially observable stochastic shortest path problem (POSSP) in which we find a trajectory that minimizes the sum of traveling and terminating cost to final absorbing states. Due to stochastic environments, the shortest trajectory varies depending on the realizations of random factors in environments.

2.4 Approximated discrete-time problem

The above continuous-time formulation captures the continuous nature of many engineering systems. However, in order to design optimal or sub-optimal control laws, the approximated discrete-time formulation is considered instead. In particular, each time step lasts Δ seconds, and thus the control horizon is $N = \frac{\mathcal{T}}{\Delta}$. Eq. 2.1 and Eq. 2.2 can be approximated as:

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k\sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{2.5}$$

$$y_k = g(x_k) + \vartheta_k, \qquad k = 0, 1, ..., N \tag{2.6}$$

where $x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$, $u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}$,$y_k \in \mathbb{R}^{n_y}$, $w_k \in \mathbb{R}^{n_w}$, and $\vartheta_k \in \mathbb{R}^{n_\vartheta}$ are samples of state, control, measurement, system noise, and measurement noise values respectively. Indeed, the discrete stochastic process x satisfies the Markov property in discrete-time domain (Definition 2.1.2). In addition, we note that in Eq. 2.5, the Ito integral of Brownian noise $w(t)$ over a duration Δ is approximated as $w_k\sqrt{\Delta}$ [32].

An information storage $I(t)$ is also approximated as a vector I_k that stores sampled measurements and sampled control signals to design a feedback control law π:

$$I_k = [y_0, y_1, ..., y_k, u_0, u_1, ..., u_{k-1}]\,. \tag{2.7}$$

As discussed in Section 2.3, we want to find a feedback control law π that is non-stationary. Thus, a discrete time control law π consists of N information-control mappings for N time steps:

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{2.8}$$

$$u_k = \mu_k(I_k). \tag{2.9}$$

Finally, the objective function and an optimal control law can be approximated by the formulae:

$$J_\pi(I_0) = E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k\big(x_k, \mu_k(I_k)\big)\Big| I_0\right], \tag{2.10}$$

$$\pi^* = arg\min_{\Pi} J_\pi(I_0). \tag{2.11}$$

In Eq. 2.10, the integral of instantaneous cost over a duration Δ is approximated as $\ell_k\big(x_k, \mu_k(I_k)\big) = \ell\big((k-1)\Delta, x_k, \mu_k(I_k)\big)\Delta$.

2.5 Statistical assumptions

Before finding a solution to this problem, to make the system tractable, we consider the following statistical assumptions of Brownian noise processes w and ϑ, and an

initial state x_0:

- w_k and ϑ_k are independent of state x_k and control u_k for all k,
- w_k is independent of ϑ_k for all k,
- w_ks are i.i.d with a Gaussian distribution $N(0, \Omega^w)$,
- ϑ_ks are i.i.d with a Gaussian distribution $N(0, \Omega^\vartheta)$,
- x_0 has a Gaussian distribution $N(x_0^-, \Lambda_0^-)$, where x_0^- is a mean value.

These assumptions of independent Gaussian distributions enable rich probability manipulations and inferences such as the Kalman filter and its variants [37]. In particular, we only need means and covariances to describe these Gaussian distributions. Thus, they provide us with computationally feasible methods to handle probability density distributions over complex transformations.

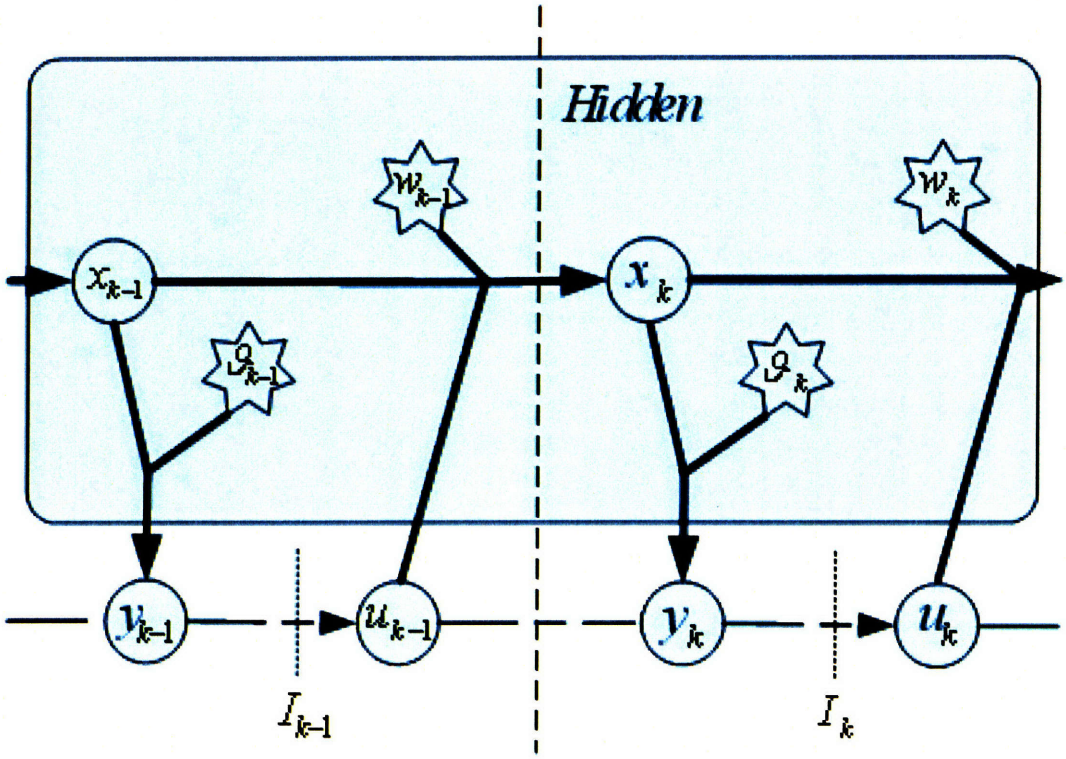


Figure 2-3: System evolution of the considered problem as a POMDP.

2.6 Optimization problem

With the above discussion, the problem under consideration can be expressed as the following constrained optimization problem:

$$\min_{\Pi} E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k) \middle| I_0\right] \tag{2.12}$$

subject to:

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k\sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{2.13}$$

$$y_k = g(x_k) + \vartheta_k, \qquad k = 0, 1, ..., N \tag{2.14}$$

$$I_k = [y_0, y_1, ..., y_k, u_0, u_1, ..., u_{k-1}], \tag{2.15}$$

$$u_k = \mu_k(I_k), \tag{2.16}$$

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{2.17}$$

$$x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, y_k \in \mathbb{R}^{n_y}, w_k \in \mathbb{R}^{n_w}, \vartheta_k \in \mathbb{R}^{n_\vartheta}, \tag{2.18}$$

$$w_k \sim N(0, \Omega^w), \vartheta_k \sim N(0, \Omega^\vartheta), x_0 \sim N(x_0^-, \Lambda_0^-), I_0 = y_0. \tag{2.19}$$

Figure 2-3 shows the evolution of this discrete-time system. As we can see, this structure resembles the well-known POMDP structure in literature. The nodes in the shaded rectangle are hidden states and noises, and the nodes outside this rectangle are the accessible measurements and control inputs.

This optimization problem is generally complex, and it is hard to find an optimal feedback control law to attain the minimum objective cost. Thus, in this research, we will look for a sub-optimal control law instead. Furthermore, we note that in this POSSP, the horizon N is unspecified. Nevertheless, for certain problems, we can exploit their structure to determine the horizon N heuristically, which we will show using the POBRM algorithm in Chapter 3.

2.7 2D model of a planar car

In this subsection, we discuss the concrete model of a planar car, which provides the dynamics and measurements of the car by a continuous-time formulation in Section 2.3. The content in this subsection is presented in detail in [30]. Thus, the brief and simplified motion and sensor models are presented here.

2.7.1 Motion model

The pose of the car, consisting of an x-y location and a heading θ, operating in a plane, is depicted in Figure 2-4. We regard the pose information of the car as the system state:

$$x(t) = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t . \tag{2.20}$$

The control input for the car has two components, namely a translational velocity v and a rotational velocity ω:

$$u(t) = \begin{bmatrix} v \\ \omega \end{bmatrix}_t . \tag{2.21}$$

We can have additional constraints to restrict these velocities in a certain range $\mathcal{U}$: $u(t) \in \mathcal{U}$.

When applying a control $u(t)$ to a state $x(t)$, the system follows the following dynamics:

$$d(x(t)) = f(x,u)dt + dw(t) \tag{2.22}$$

$$= \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix} dt + dw(t). \tag{2.23}$$

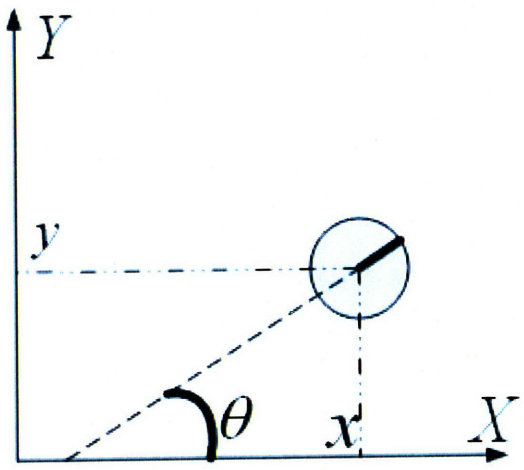


Figure 2-4: Car pose in the X-Y plane.

2.7.2 Feature-based map and sensor model

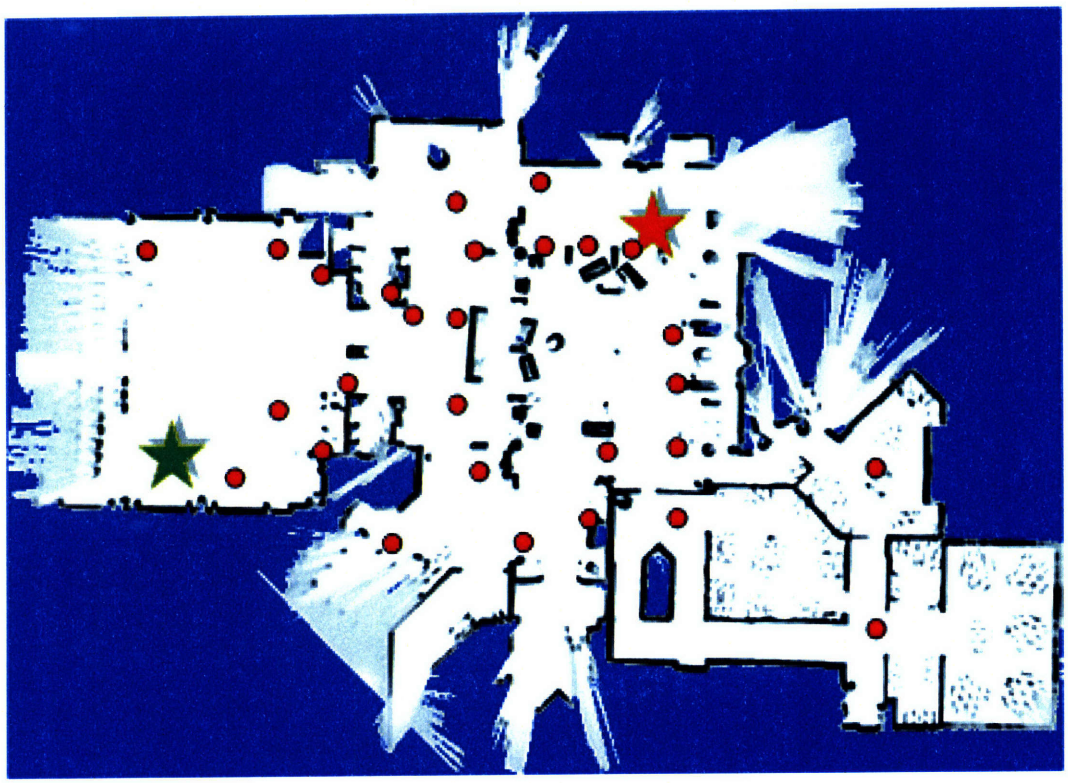


Figure 2-5: An example of a feature-based map.

Figure 2-5 shows an example of the typical map of an area in which the car may operate. Black objects are obstacles, and red dots are landmarks, or features. The green star is a starting location, and the red star is a destination. The area contains rich information for the car such as corners of buildings, walls, doors, and windows. These objects are called landmarks in a feature-based map. With a given feature-based map, the POBRM algorithm will exploit the map structure to solve the optimization problem in Section 2.6.

The car is equipped with sensors to work in the area. For a certain landmark locating at $[m_x \quad m_y]^T$ in that area, the car can sense the distance and relative bearing to the landmark if it is within the visibility region of the car. We assume that when the car senses the landmark, it knows the landmark's coordinates. Depending on the car's location, at any instant of time, the car can receive no measurement, or a list different measurements from several landmarks. The measurement model is described by

$$if \quad \sqrt{(m_x - x)^2 + (m_y - y)^2} \leq R_m : \tag{2.24}$$

$$y(t, m) = g(x, m) + \vartheta(t) \tag{2.25}$$

$$= \begin{bmatrix} \sqrt{(m_x - x)^2 + (m_y - y)^2} \\ atan2(m_y - y, m_x - x) - \theta \end{bmatrix} + \vartheta(t), \tag{2.26}$$

where R_m is the range of visibility of the car for that landmark.

2.7.3 Cost model

We want to find an optimal trajectory from a starting location to a destination with the final state x_G. The final stage cost function and instantaneous cost function are

$$h(x(\mathcal{T})) = (x(\mathcal{T}) - x_G)^T Q(\mathcal{T})(x(\mathcal{T}) - x_G), \tag{2.27}$$

$$\ell(t, x(t), u(t)) = u(t)^T R(t) u(t), \tag{2.28}$$

$$Q(\mathcal{T}) \succ 0, R(t) \succ 0. \tag{2.29}$$

The weight matrices $Q(\mathcal{T}), R(t)$ are symmetric positive definite. Using these cost functions, we penalize large deviations from the final state x_G and prefer trajectories with low energy consumption that reach the destination. Moreover, there is a trade-off between these cost components by setting different weight matrices $Q(\mathcal{T})$ and $R(t)$.

2.8 3D model of a helicopter

In this subsection, we show how the optimization problem in Section 2.6 can be used to control the simplified dynamics and measurements of a six-degree-of-freedom (6-DOF) quad-helicopter. For other complex models of aerospace vehicles and quad-helicopters, readers can refer to [39] and [9].

2.8.1 Motion model

To define the dynamics of the helicopter, we have to work in the North-East-Down (NED) inertial coordinate system, and the body coordinate system, as shown in Figure 2-6. The NED coordinate system embeds its X_E and Y_E axes into a horizontal plane and points Z_E axis downward according to the right hand side rule with the fixed origin. The body coordinate system is aligned with the helicopter triad with the origin at the center of gravity (COG) of the vehicle. The X_B axis points through the noise of the vehicle, and the Y_B axis points out to the right of the X_B axis. Finally, the Z_B axis pointing downward completes the coordinate system.

The x-y-z locations are defined in the NED coordinate system, while the yaw ψ, pitch θ, and roll ϕ angles are defined as rotational angles from the NED coordinate system to the body coordinate system. We define the state of the helicopter as follows:

$$x(t) = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \psi & \theta & \phi \end{bmatrix}_t^T, \tag{2.30}$$

where $[\dot{x} \quad \dot{y} \quad \dot{z}]^T$ is the velocity of the helicopter.

Controlling the 6-DOF helicopter involves the details of translational dynamics

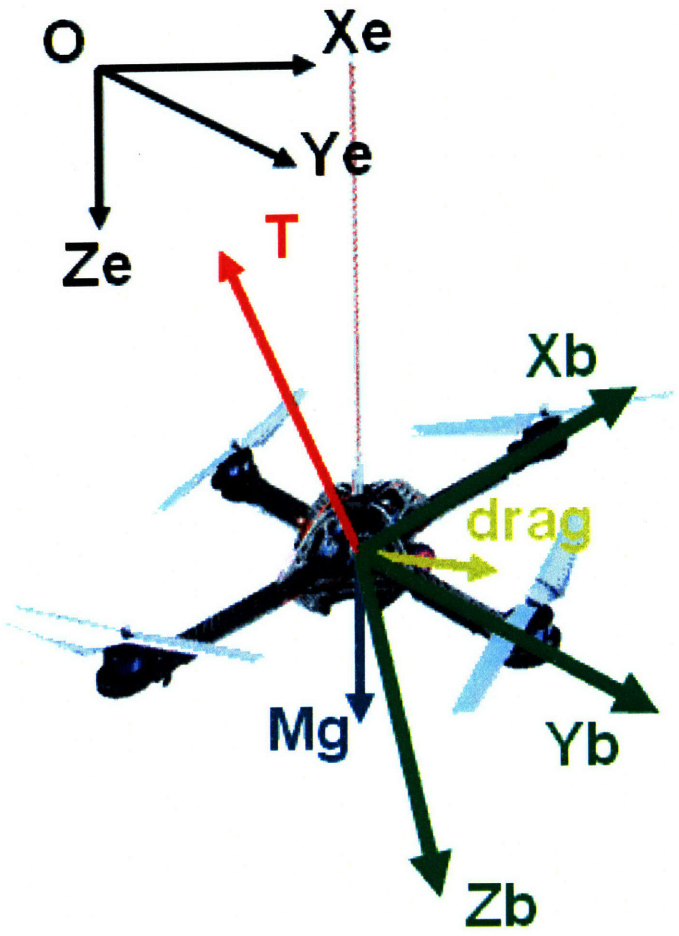


Figure 2-6: Free body diagram of a quad-helicopter.

and attitude dynamics. On one hand, translational dynamics describes trajectories of $x-y-z$ points under external forces as recorded by an observer. The Newton's law is a key tool to obtain translational dynamics. On the other hand, attitude dynamics describes how the vehicle rotates under external forces. The Euler's law is a main tool to obtain the dynamics of Euler angles ψ, θ, and ϕ.

In this research, to keep the model simple, we assume that angle velocity can be controlled directly. Thus, we concern only with the Newton's law but not the Euler's law under external forces. Moreover, we also assume that the thrust generated does not depend on voltage power left on the helicopter. The control input is expressed as

follows

$$u(t) = \begin{bmatrix} T & \dot{\psi} & \dot{\theta} & \dot{\phi} \end{bmatrix}_t^T, \tag{2.31}$$

$$T \geq T_{min} \geq 0, \tag{2.32}$$

where T is nonnegative thrust with minimum thrust T_{min}, $\begin{bmatrix} \dot{\psi} & \dot{\theta} & \dot{\phi} \end{bmatrix}^T$ is rotational velocity. The transform matrix to rotate from the body coordinate system to the NED coordinate system is given by

$$[T]^{EB} = [-\psi]_{Z_B} [-\theta]_{Y_B} [-\phi]_{X_B} \tag{2.33}$$

$$= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.34}$$

$$= \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\theta + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi + \cos\psi\sin\theta\cos\phi \\ \sin\psi\cos\theta & \cos\psi\cos\theta + \sin\psi\sin\theta\sin\phi & -\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}. \tag{2.35}$$

As shown in Figure 2-6, given M is the weight of the helicopter, we assume that an applied thrust is perpendicular to the helicopter body plane going through its COG. In addition, the velocity of the helicopter is small enough so that the drag force is proportional by a coefficient k and in an opposite direction to the translational velocity of the helicopter. By the Newton's law, we have:

$$[T]^{EB} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} T + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} Mg - k \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = M \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} \tag{2.36}$$

$$\begin{bmatrix} T(-\sin\psi\sin\phi - \cos\psi\sin\theta\cos\phi) \\ T(\cos\psi\sin\phi - \sin\psi\sin\theta\cos\phi) \\ -T\cos\theta\cos\phi \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ Mg \end{bmatrix} + \begin{bmatrix} -k\dot{x} \\ -k\dot{y} \\ -k\dot{z} \end{bmatrix} = \begin{bmatrix} M\ddot{x} \\ M\ddot{y} \\ M\ddot{z} \end{bmatrix} \tag{2.37}$$

$$\begin{bmatrix} \frac{T}{M}(-\sin\psi\sin\phi - \cos\psi\sin\theta\cos\phi) - \frac{k}{M}\dot{x} \\ \frac{T}{M}(\cos\psi\sin\phi - \sin\psi\sin\theta\cos\phi) - \frac{k}{M}\dot{y} \\ -\frac{T}{M}\cos\theta\cos\phi - \frac{k}{M}\dot{z} + g \end{bmatrix} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}. \tag{2.38}$$

Thus, the helicopter dynamics are described by

$$d(x(t)) = f(x,u)dt + dw(t) \tag{2.39}$$

$$= \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \frac{T}{M}(-\sin\psi\sin\phi - \cos\psi\sin\theta\cos\phi) - \frac{k}{M}\dot{x} \\ \frac{T}{M}(\cos\psi\sin\phi - \sin\psi\sin\theta\cos\phi) - \frac{k}{M}\dot{y} \\ -\frac{T}{M}\cos\theta\cos\phi - \frac{k}{M}\dot{z} + g \\ \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} dt + dw(t). \tag{2.40}$$

2.8.2 Sensor model

Similar to the 2D case, the feature-based map of an operating area is given. The helicopter is equipped with sensors to measure a distance to a landmark, and its translational velocity when the landmark is within its visibility. Moreover, as the pitch and roll angles are local knowledge of the helicopter, we assume that the helicopter can sense the pitch and roll angles at any time using gravity and a three-axis accelerometer. In addition, we assume that the helicopter can estimate the yaw angle. These two components of measurement are described by

$$if \quad \sqrt{(m_x - x)^2 + (m_y - y)^2 + (m_z - z)^2} \leq R_m : \tag{2.41}$$

$$y_1(t,m) = g_1(x,m) + \vartheta_1(t) \tag{2.42}$$

$$\tag{2.43}$$

$$= \begin{bmatrix} \sqrt{(m_x - x)^2 + (m_y - y)^2 + (m_z - z)^2} \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} + \vartheta_1(t), \tag{2.44}$$

and

$$y_2(t) = g_2(x) + \vartheta_2(t) \tag{2.45}$$

$$= \begin{bmatrix} \psi \\ \theta \\ \phi \end{bmatrix} + \vartheta_2(t), \tag{2.46}$$

2.8.3 Cost model

Similar to the 2D car, we also find an optimal path from a starting location to a destination with the final state x_G. The final stage cost function and instantaneous cost function are

$$h(x(T)) = (x(T) - x_G)^T Q(T)(x(T) - x_G), \tag{2.47}$$

$$\ell(t, x(t), u(t)) = u(t)^T R(t) u(t), \tag{2.48}$$

$$Q(T) \succ 0, R(t) \succ 0. \tag{2.49}$$

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

POBRM Algorithm

In this chapter, we discuss in detail how to solve the optimization problem defined by Eq. 2.12 to Eq. 2.19 in Section 2.6. We start with the framework of dynamic programming (DP). The overview of the extended Kalman filter follows. Then, the iterative Linear Quadratic Gaussian (iLQG) algorithm is presented. After that, the offline and online phases of the POBRM algorithm are discussed. We attempt to provide a preliminary analysis of the POBRM algorithm. There are several sources of error that affect the accuracy of a returned sub-optimal solution. Next, we argue that, under special assumptions, it is possible to bound the optimal solution.

3.1 The framework of DP

The basic framework of dynamic programming, in which decisions are made in stages, is well studied in [3]. We notice that the formulated POSSP in Section 2.6 has a suitable structure to be solved by DP. The world of DP related algorithms such as MDP, POMDP, and Neuro-DP, as reviewed in Chapter 2, is vast. Hereafter, the usual case of fully observable DP is presented, and the extended partially observable DP follows.

3.1.1 Fully observable case

The decision making problem in a fully observable environment is described by

$$\min_{\Pi} E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k)\Big| x_0\right] \tag{3.1}$$

subject to:

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k\sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{3.2}$$

$$u_k = \mu_k(x_k), \tag{3.3}$$

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{3.4}$$

$$x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, w_k \in \mathbb{R}^{n_w}, \tag{3.5}$$

$$w_k \sim N(0, \Omega^w). \tag{3.6}$$

The notations in Eq. 3.1 to Eq. 3.6 have the same meanings as their counterparts in Eq. 2.12 to Eq. 2.19. However, this formulation is much simpler as it formulates an MDP. First, all states of the system are directly accessible, thus no measurement output is needed. Second, an admissible feedback control input at any instant of time is a function of a state. Third, an objective function is evaluated given an observable starting state.

The DP framework is based on the following lemma called the principle of optimality proposed by Bellman.

Lemma 3.1.1. *(Principle of optimality [3]) Let $\pi^* = \{\mu_0^*, \mu_1^*, ..., \mu_{N-1}^*\}$ be an optimal policy for the above problem, and assume that when using π^*, a given state x_i occurs at time i with positive probability. Consider the subproblem whereby we are at x_i and minimize the objective function from time i to time N*

$$E\left[h(x_N) + \sum_{k=i}^{N-1} \ell_k(x_k, u_k)\Big| x_i\right].$$

Then the truncated policy $\{\mu_i^*, \mu_{i+1}^*, ..., \mu_{N-1}^*\}$ is optimal for this subproblem.

Proof. Since π^* is an optimal policy, the cost objective J^* induced by π^* is the smallest. If the truncated policy is not optimal for the subproblem, let $\{\mu_i', \mu_{i+1}', ..., \mu_{N-1}'\}$ be an optimal sub-policy. Then the new policy $\pi' = \{\mu_0^*, \mu_1^*, ..., \mu_{i-1}^*, \mu_i', \mu_{i+1}', ..., \mu_{N-1}'\}$ yields a smaller value of the original objective function than J^*, which is a contradiction. □

Using Lemma 3.1.1, it can be proved by induction that the optimal cost for the basic fully observable DP problem can be computed backward as in the following theorem [3]:

Theorem 3.1.1. *Denote:*

$$J_\pi(x_0) = E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k)\middle| x_0\right],$$
$$J^*(x_0) = J_{\pi^*}(x_0).$$

For every initial state x_0, the optimal cost $J^*(x_0)$ is equal to $J_0(x_0)$, which is given by the following formulas for k from $N-1$ down-to 0:

$$J_N(x_N) = h(x_N), \tag{3.7}$$
$$J_k(x_k) = \min_{u_k \in \mathcal{U}_k(x_k)} E_{w_k}\left[\ell_k(x_k, u_k) + J_{k+1}(x_{k+1})\middle| x_k, u_k\right], \quad k = N-1, N-2, ..., 1, 0. \tag{3.8}$$

Furthermore, let u_k^* be a minimizer of the Eq. 3.8, and we assign $\mu^*(x_k) = u_k^*$ for every x_k and k, then the policy $\pi^* = \{\mu_0^*, ..., \mu_{N-1}^*\}$ is optimal.

In Eq. 3.8, we have a minimization problem over a feasible set $\mathcal{U}_k(x_k)$, which is a subset of $\mathcal{U}$. This feasible set depends on the current state x_k because only control inputs in $\mathcal{U}$ that produce x_{k+1} in $\mathcal{X}$ are acceptable.

3.1.2 Partially observable case

Let us turn our attention back to the DP framework in a partially observable environment. For the purpose of clarity, we restate the formulation here:

$$\min_{\Pi} E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k)\middle| I_0\right] \tag{3.9}$$

subject to:

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k\sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{3.10}$$

$$y_k = g(x_k) + \vartheta_k, \qquad k = 0, 2, ..., N \tag{3.11}$$

$$I_k = [y_0, y_1, ..., y_k, u_0, u_1, ..., u_{k-1}], \tag{3.12}$$

$$u_k = \mu_k(I_k), \tag{3.13}$$

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{3.14}$$

$$x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, y_k \in \mathbb{R}^{n_y}, w_k \in \mathbb{R}^{n_w}, \vartheta_k \in \mathbb{R}^{n_\vartheta}, \tag{3.15}$$

$$w_k \sim N(0, \Omega^w), \vartheta_k \sim N(0, \Omega^\vartheta), x_0 \sim N(x_0^-, \Lambda_0^-), I_0 = y_0. \tag{3.16}$$

The Theorem 3.1.1 cannot be applied directly to this formulation due to the inaccessibility of states x. However, it is possible to reformulate the problem to become fully observable with respect to new information states I, control inputs u, and random disturbances y:

$$I_0 = y_0, \tag{3.17}$$

$$I_{k+1} = [I_k, u_k, y_{k+1}], \quad k = 1, 2, ..., N-1. \tag{3.18}$$

To see this, we first show that I_k satisfies the Markov property.

Lemma 3.1.2. *With the dynamics of variables as defined in Eq. 3.9 to Eq. 3.18, the discrete stochastic process I_k obeys the Markov property.*

Proof. From Eq. 3.18, we note that I_k is part of I_{k+1} for all k, thus

$$P\left[I_{k-1}, u_{k-1}, ..., I_0, u_0 \middle| I_k, u_k, ..., I_0, u_0\right] = 1.$$

Therefore the following conditional probability holds:

$$\begin{aligned} P\left[I_k, u_k, y_{k+1} \middle| I_k, u_k, ..., I_0, u_0\right] &= \frac{P\left[I_k, u_k, y_{k+1} \middle| I_k, u_k\right]}{P\left[I_{k-1}, u_{k-1}, ..., I_0, u_0 \middle| I_k, u_k, ..., I_0, u_0\right]} \\ &= P\left[I_k, u_k, y_{k+1} \middle| I_k, u_k\right]. \end{aligned}$$

Again, From the dynamics of I_{k+1} in Eq. 3.18, it follows that

$$P\left[I_{k+1} \middle| I_k, u_k\right] = P\left[I_{k+1} \middle| I_k, u_k, ..., I_0, u_0\right].$$

□

Corollary 3.1.1. *Let X be any random variable:*

$$E\left[X \middle| I_i, I_j\right] = E\left[X \middle| I_i\right], \text{ for } j \leq i.$$

This corollary states that the expectation of a random variable depends only on the current information vector but not past information vectors. Furthermore, we have the following facts:

Fact 3.1.1. *Two common iterative expectation rules:*

- *Let X, Y, Z be random variables:*

$$E\left[X \middle| Y\right] = E\left[E[X|Y, Z] \middle| Y\right]. \tag{3.19}$$

- *Let X, Y, Z be random variables:*

$$E\left[E[X|Y] + Z \middle| Y\right] = E\left[X + Z \middle| Y\right]. \tag{3.20}$$

Using these corollary and facts, the following theorem shows how the DP framework works in a partially observable environment.

Theorem 3.1.2. *Denote:*

$$J_\pi(I_0) = E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k)\Big| I_0\right],$$
$$J^*(I_0) = J_{\pi^*}(I_0).$$

For every initial information I_0, the optimal cost $J^*(I_0)$ is equal to $J_0(I_0)$, which is given by the following formulas for k from $N-1$ down-to 0:

$$J_N(I_N) = E_{x_N}\left[h(x_N)\Big| I_N\right], \tag{3.21}$$
$$J_k(I_k) = \min_{u_k \in \mathcal{U}_k(I_k)} E_{x_k, w_k, y_{k+1}}\left[\ell_k(x_k, u_k) + J_{k+1}(I_{k+1})\Big| I_k, u_k\right], \quad k = N-1, ..., 0. \tag{3.22}$$

Furthermore, let u_k^* be a minimizer of the Eq. 3.22, and we assign $\mu^*(I_k) = u_k^*$ for every I_k and k, then the policy $\pi^* = \{\mu_0^*, ..., \mu_{N-1}^*\}$ is optimal.

Proof. First, we rewrite the objective function in terms of information state I_k as in a fully observable case:

$$J_\pi(I_0) = E_{x_{0,x_{1..N}}}\left[h(x_N) + \sum_{i=0}^{N-1} \ell_k(x_k, u_k)|I_0\right] \tag{3.23}$$
$$= E_{I_{1..N}}\left\{E_{x_N}\left[h_N(x_N)\Big| I_N\right] + \sum_{i=0}^{N-1} E_{x_k}\left[\ell_k(x_k, u_k)\Big| I_k, u_k\right]\Big| I_0\right\} \tag{3.24}$$
$$= E_{I_{1..N}}\left[\widetilde{h_N}(I_N) + \sum_{i=0}^{N-1} \widetilde{\ell_k}(I_k, u_k)|I_0\right], \tag{3.25}$$

where the first equality is obtained using the iterative expectation rule in Fact 3.19 and reducing the inner conditional expectation using the Markov property in Corollary

3.1.1 with shorthands:

$$\widetilde{h_N}(I_N) = E_{x_N}\left[h_N(x_N)\middle|I_N\right], \tag{3.26}$$

$$\widetilde{\ell_k}(I_k, u_k) = E_{x_k}\left[\ell_k(x_k, u_k)\middle|I_k, u_k\right]. \tag{3.27}$$

Therefore, now we can express the cost function as the newly set-up fully observable system:

$$J_\pi(I_0) = E_{I_{1..N}}\left[\widetilde{h_N}(I_N) + \sum_{i=0}^{N-1}\widetilde{\ell_k}(I_k, u_k)\middle|I_0\right]. \tag{3.28}$$

Hence, the DP framework in Theorem 3.1.1 with respect to I_k is applied to have $J^*(I_0) = J_0(I_0)$ from the following system of equations:

$$J_N(I_N) = \widetilde{h_N}(I_N) = E_{x_N}\left[h_N(x_N)\middle|I_N\right] \tag{3.29}$$

$$J_k(I_k) = \min_{u_k \in \mathcal{U}_k(I_k)} E_{y_{k+1}}\left[\widetilde{\ell_k}(I_k, u_k) + J_{k+1}(I_{k+1})\middle|I_k, u_k\right] \tag{3.30}$$

$$= \min_{u_k \in \mathcal{U}_k(I_k)} E_{y_{k+1}}\left\{E_{x_k}\left[\ell_k(x_k, u_k)\middle|I_k, u_k\right] + J_{k+1}(I_{k+1})\middle|I_k, u_k\right\} \tag{3.31}$$

$$= \min_{u_k \in \mathcal{U}_k(I_k)} E_{y_{k+1}, x_k, w_k}\left[\ell_k(x_k, u_k) + J_{k+1}(I_{k+1})\middle|I_k, u_k\right], \tag{3.32}$$

where the last equality uses the expectation rule in Fact 3.20.

In addition, an optimal policy can be constructed accordingly. □

3.2 Extended Kalman filter

The above backward recursion system in Theorem 3.1.2 is generally complex to solve as the dimension of states I increases over time. So, out of all information encapsulated inside I_k, we would like to keep only the useful information. This is where the assumptions of Gaussian distributions in Section 2.5 come into place. In this case, we try to keep conditional means $\hat{x}_k = E[x_k|I_k]$ and conditional covariances $\Lambda_k = Var[x_k|I_k]$ using the Extended Kalman filter (EKF). The EKF has been well studied in literature [30] [23], as an approximation to the Kalman filter [23] in

non-linear domains by linearizing the dynamics of systems and measurements. The summary of how the EKF works is described in the following theorem.

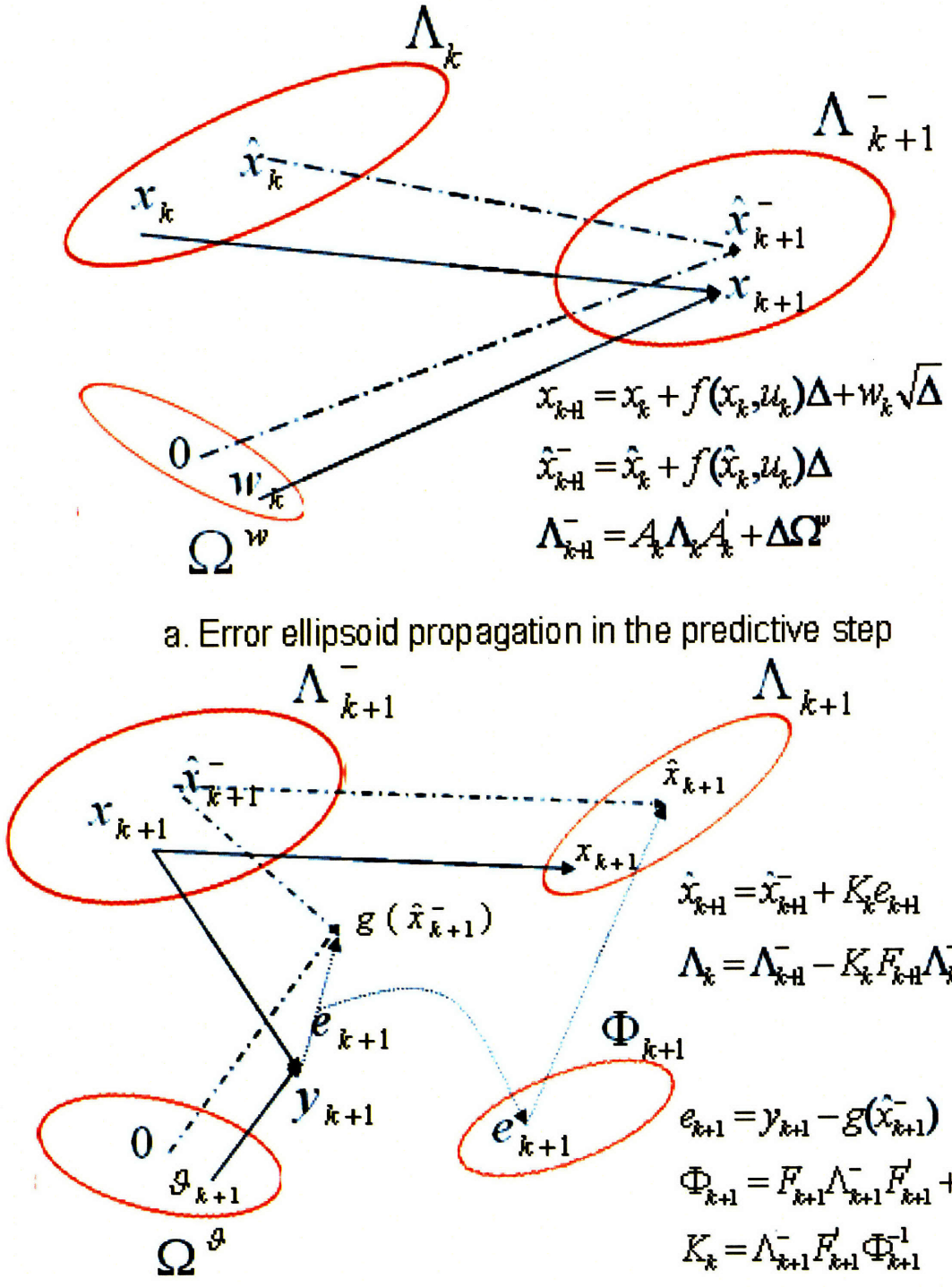


Figure 3-1: The extended Kalman filter.

Theorem 3.2.1. *(The extended Kalman filter) Assume that the system together with its measurement output is described by*

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k\sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{3.33}$$

$$y_k = g(x_k) + \vartheta_k, \qquad k = 0, 1, ..., N. \tag{3.34}$$

Assume that a control input u_k at time k is given. The conditional mean $\hat{x}_{k+1}$ and conditional covariance Λ_{k+1} of a system state at time $k+1$ after observing a measurement y_k at time k are computed recursively via the following two steps:

Predictive step:

$$\hat{x}_{k+1}^- = \hat{x}_k + f(\hat{x}_k, u_k)\Delta, \tag{3.35}$$

$$A_k = I + \frac{\partial f}{\partial x}\Big|_{\hat{x}_k, u_k}\Delta, \tag{3.36}$$

$$\Lambda_{k+1}^- = A_k\Lambda_k A_k' + \Delta\Omega^w. \tag{3.37}$$

Update step:

$$F_{k+1} = \frac{dg}{dx}\Big|_{\hat{x}_{k+1}^-}, \tag{3.38}$$

$$K_k = \Lambda_{k+1}^- F_{k+1}'(F_{k+1}\Lambda_{k+1}^- F_{k+1}' + \Omega^v)^{-1}, \tag{3.39}$$

$$\hat{x}_{k+1} = \hat{x}_{k+1}^- + K_k\big(y_{k+1} - g(\hat{x}_{k+1}^-)\big), \tag{3.40}$$

$$\Lambda_{k+1} = \Lambda_{k+1}^- - K_k F_{k+1}\Lambda_{k+1}^-. \tag{3.41}$$

Information form: *Alternatively, we can maintain the information form of the EKF, which stores and updates the inverses of conditional covariances and information vectors ν as:*

$$\nu_{k+1}^- = (\Lambda_{k+1}^-)^{-1}\hat{x}_{k+1}^-, \tag{3.42}$$

$$(\Lambda_{k+1}^-)^{-1} = (A_k\Lambda_k A_k' + \Delta\Omega^w)^{-1}, \tag{3.43}$$

$$\nu_{k+1} = F'_{k+1}(\Omega^v)^{-1}y_{k+1} + \nu^-_{k+1}, \tag{3.44}$$

$$\Lambda^{-1}_{k+1} = (\Lambda^-_{k+1})^{-1} + F'_{k+1}(\Omega^v)^{-1}F_{k+1}. \tag{3.45}$$

The interpretation of the EKF is depicted graphically in Figure 3-1. As we can see, the ellipses represent the contour of equal probability around the means. The dashed lines depict the dynamics of the mean values, while the solid lines represent the actual values of random variables.

The following lemma and Theorem provide another way to update covariance matrices.

Lemma 3.2.1. *(Inversion lemma)*

$(A + BC^{-1})^{-1} = (ACC^{-1} + BC^{-1})^{-1} = C(AC + B)^{-1}$.

Theorem 3.2.2. *If a conditional covariance is factored as $\Lambda_k = \mathcal{B}_k\mathcal{C}_k^{-1}$, the predictive conditional covariance in Eq. 3.37, and the update conditional covariance in Eq. 3.41 can be factored as*

$$\Lambda^-_{k+1} = \mathcal{B}^-_{k+1}(\mathcal{C}^-_{k+1})^{-1}, \tag{3.46}$$

$$\Lambda_{k+1} = \mathcal{B}_{k+1}\mathcal{C}^{-1}_{k+1}, \tag{3.47}$$

where

- *$\mathcal{B}^-_{k+1}$ and $\mathcal{C}^-_{k+1}$ are linear functions of $\mathcal{B}_k$ and $\mathcal{C}_k$*
- *$\mathcal{B}_{k+1}$ and $\mathcal{C}_{k+1}$ are linear functions of $\mathcal{B}^-_{k+1}$ and $\mathcal{C}^-_{k+1}$.*

Proof. From $\Lambda_k = \mathcal{B}_k\mathcal{C}_k^{-1}$,using the inversion lemma, we have:

$$\Lambda^-_{k+1} = A_k\Lambda_k A'_k + \Delta\Omega^w \tag{3.48}$$

$$= A_k\mathcal{B}_k\mathcal{C}_k^{-1}A'_k + \Delta\Omega^w \tag{3.49}$$

$$= \left(\left(\left(A_k\mathcal{B}_k\right)\left((A'_k)^{-1}\mathcal{C}_k\right)^{-1} + \Delta\Omega^w\right)^{-1}\right)^{-1} \tag{3.50}$$

$$= \left(\left((A'_k)^{-1}\mathcal{C}_k\right)\left(\Delta\Omega^w(A'_k)^{-1}\mathcal{C}_k + A_k\mathcal{B}_k\right)^{-1}\right)^{-1} \tag{3.51}$$

$$= \left(\Delta\Omega^w(A'_k)^{-1}\mathcal{C}_k + A_k\mathcal{B}_k\right)\left((A'_k)^{-1}\mathcal{C}_k\right)^{-1} \tag{3.52}$$

$$= \mathcal{B}^-_{k+1}(\mathcal{C}^-_{k+1})^{-1}, \tag{3.53}$$

where

$$\begin{bmatrix}\mathcal{B}^-_{k+1}\\ \mathcal{C}^-_{k+1}\end{bmatrix} = \begin{bmatrix}A_k & \Delta\Omega^w(A'_k)^{-1}\\ 0 & (A'_k)^{-1}\end{bmatrix}\begin{bmatrix}\mathcal{B}_k\\ \mathcal{C}_k\end{bmatrix} = [\xi^-_{k+1}]\begin{bmatrix}\mathcal{B}_k\\ \mathcal{C}_k\end{bmatrix} \tag{3.54}$$

Using the information form and applying the inversion lemma again, we have:

$$\Lambda_{k+1} = \left((\Lambda^-_{k+1})^{-1} + F'_{k+1}(\Omega^v)^{-1}F_{k+1}\right)^{-1} \tag{3.55}$$

$$= \left(\mathcal{C}^-_{k+1}(\mathcal{B}^-_{k+1})^{-1} + F'_{k+1}(\Omega^v)^{-1}F_{k+1}\right)^{-1} \tag{3.56}$$

$$= \mathcal{B}^-_{k+1}\left(F'_{k+1}(\Omega^v)^{-1}F_{k+1}\mathcal{B}^-_{k+1} + \mathcal{C}^-_{k+1}\right)^{-1} \tag{3.57}$$

$$= \mathcal{B}_{k+1}\mathcal{C}^{-1}_{k+1}, \tag{3.58}$$

where

$$\begin{bmatrix}\mathcal{B}_{k+1}\\ \mathcal{C}_{k+1}\end{bmatrix} = \begin{bmatrix}I & 0\\ F'_{k+1}(\Omega^v)^{-1}F_{k+1} & I\end{bmatrix}\begin{bmatrix}\mathcal{B}^-_{k+1}\\ \mathcal{C}^-_{k+1}\end{bmatrix} = [\xi_{k+1}]\begin{bmatrix}\mathcal{B}^-_{k+1}\\ \mathcal{C}^-_{k+1}\end{bmatrix}. \tag{3.59}$$

Obviously, $[\xi^-_{k+1}]$ and $[\xi_{k+1}]$ are linear operators whose values depends only on control inputs and received measurements. □

3.3 iLQG algorithm

This section focuses on the main tool, called the iLQG algorithm, to find a sub-optimal control policy for subproblems in POBRM. The iLQG algorithm is proposed by Weiwei Li and Todorov [15], which is relevant to the work by Tse [33] . We will present the iLQG algorithm with some slight changes from the Li's version. The DP framework for partially observable environments (Theorem 3.1.2) is used to obtain a

sub-optimal control policy with the assistance of the EKF (Theorem 3.2.1).

In the discussion of the iLQG algorithm, we keep the cost function in the general form $J_\pi(I_0) = E\left[h_N(x_N) + \sum_{i=0}^{N-1} \ell_k(x_k, u_k) \middle| I_0\right]$ instead of the proposed functions for a planar car or a quad-helicopter in Section 2.7 and Section 2.8. Moreover, the form of $h_N(x_N)$ and $\ell_k(x_k, u_k)$ may not necessarily be quadratic. In fact, we will formulate suitable forms for these functions to solve different subproblems in the POBRM algorithm.

With non-linear system dynamics and possibly a non-quadratic cost function, the iLQG algorithm linearizes the system dynamics and approximates the cost function up to second order around a series of nominal trajectories. The first nominal trajectory is arbitrarily guessed, and the next better trajectory is calculated until a convergence criteria is met. In addition, we will obtain a sub-optimal locally-feedback control law around the vicinity of a sub-optimal trajectory. We construct nominal trajectories as follows:

Definition 3.3.1. *Assume that a sequence of open-loop control inputs $\bar{u}_k$ is given, a nominal trajectory $\bar{x}_k$ is generated by simulating the dynamics without noise using control inputs $\bar{u}_k$ from a given $\bar{x}_0$ that is the mean of an initial state distribution:*

$$\bar{x}_{k+1} = f(\bar{x}_k, \bar{u}_k), \quad k = 0, 1, .., N-1. \tag{3.60}$$

The following lemma presents how to linearize the dynamics and approximate the cost function around a nominal trajectory.

Lemma 3.3.1. *If a nominal trajectory $\bar{x}_k$ is obtained, and function $\ell_k(x_k, u_k)$ is separable for variables x_k and u_k, which means $\frac{\partial^2 \ell_k}{\partial x \partial u} = 0$, the system dynamics and cost function can be approximated as:*

$$\delta x_k = x_k - \bar{x}_k, \tag{3.61}$$

$$\delta u_k = u_k - \bar{u}_k, \tag{3.62}$$

$$\delta x_{k+1} = A_k \delta x_k + B_k \delta u_k + C_k w_k, \tag{3.63}$$

$$A_k = I + \frac{\partial f}{\partial x}\Big|_{\bar{x}_k, \bar{u}_k} \Delta, \tag{3.64}$$

$$B_k = \frac{\partial f}{\partial u}\Big|_{\bar{x}_k, \bar{u}_k} \Delta, \tag{3.65}$$

$$C_k = \sqrt{\Delta}, \tag{3.66}$$

$$h(x_N) = \delta x_N' Q_N \delta x_N + q_N' \delta x_N + p_N = h(\delta x_N), \tag{3.67}$$

$$Q_N = \frac{d^2 h}{2dx^2}\Big|_{\bar{x}_N}, \tag{3.68}$$

$$q_N = \frac{dh}{dx}\Big|_{\bar{x}_N}, \tag{3.69}$$

$$p_N = h(\bar{x}_N), \tag{3.70}$$

$$\ell_k(x_k, u_k) = \delta x_k' Q_k \delta x_k + q_k' \delta x_k + \delta u_k' T_k \delta u_k + t_k' \delta u_k + p_k = \ell_k(\delta x_k, \delta u_k), \tag{3.71}$$

$$Q_k = \frac{\partial^2 \ell_k}{2\partial x^2}\Big|_{\bar{x}_N, \bar{u}_k}, \tag{3.72}$$

$$q_k = \frac{\partial \ell_k}{\partial x}\Big|_{\bar{x}_N, \bar{u}_k}, \tag{3.73}$$

$$T_k = \frac{\partial^2 \ell_k}{2\partial u^2}\Big|_{\bar{x}_N, \bar{u}_k}, \tag{3.74}$$

$$t_k = \frac{\partial \ell_k}{\partial u}\Big|_{\bar{x}_N, \bar{u}_k}, \tag{3.75}$$

$$p_k = \ell_k(\bar{x}_k, \bar{u}_k). \tag{3.76}$$

Proof. It is straightforward to check that the above approximation results from expanding the Taylor's series up to second order around the nominal trajectory $\bar{x}_k$ and nominal control inputs $\bar{u}_k$. □

Variables δx_k and δu_k represent the deviation from nominal states $\bar{x}_k$ and nominal control inputs $\bar{u}_k$ of actual random variables x_k and u_k. In addition, we rewrite functions h and ℓ_k as functions of δx_k and δu_k. Within a *tube* along a nominal trajectory, we solve a *local* optimization problem using Theorem 3.1.2 under an assumption of certainty equivalence (CE). Hereafter, we first show how to design a controller without any state or control constraints in Theorem 3.3.1, then we handle state and control constraints in Theorem 3.3.2.

Definition 3.3.2. *(Certainty equivalence principle [3]) If an optimal policy is unaf-*

fected when disturbances are replaced by their means, we say that certainty equivalence holds.

Definition 3.3.3. *(Certainty equivalent controller [3]) The certainty equivalent controller (CEC) is a sub-optimal control scheme that is inspired by linear quadratic control Theorem. It applies at each stage the control that would be optimal if the uncertain quantities were fixed at some typical values. That is, the controller assumes that the CE principle holds.*

Theorem 3.3.1. *Given an* ***unconstrained*** *optimization problem described in Eq. 2.12 to Eq. 2.19 ($\mathcal{X} = \mathbb{R}^{n_x}, \mathcal{U} = \mathbb{R}^{n_u}$), a sub-optimal CE-type control law around a nominal trajectory is generated using the following backward recursive equations:*

$$u_k = \mu_k(I_k) = \bar{u}_k + l_k + L_k(\hat{x}_k - \bar{x}_k), \tag{3.77}$$

where

$$H_k = T_k + B_k' S_{k+1} B_k, \tag{3.78}$$

$$g_k = (t_k' + s_{k+1}{}' B_k)', \tag{3.79}$$

$$G_k = 2(A_k' S_{k+1} B_k)', \tag{3.80}$$

$$l_k = -\frac{1}{2} H_k g_k, \tag{3.81}$$

$$L_k = -\frac{1}{2} H_k G_k, \tag{3.82}$$

$$P_k = L_k' H_k L_k + G_k{}' L_k, \tag{3.83}$$

$$S_k = Q_k + A_k' S_{k+1} A_k + P_k, \tag{3.84}$$

$$s_k = (q_k' + s_{k+1}{}' A_k + l_k' G_k + 2 l_k' H_k L_k + g_k{}' L_k)', \tag{3.85}$$

$$r_k = r_{k+1} + p_k + tr(C_k' S_{k+1} C_k \Omega_k^w) + l_k' H_k l_k + g_k{}' l_k - tr(P_k \Lambda_k), \tag{3.86}$$

$$S_N = Q_N, s_N = q_N, r_N = p_N, \tag{3.87}$$

$$\hat{x}_k = E[x_k | I_k], \tag{3.88}$$

$$\Lambda_k = Var[x_k | I_k]. \tag{3.89}$$

In the above equations, l_k is the refined open-loop component, and L_k is the feedback gain matrix for the time index k. Furthermore, the weight matrix S_k represents the effort, which the controller should spend to reach the destination, due to the deviation from $\bar{x}_k$ in the subproblem formed from the time index k to the final time index N. In addition, the term r_k represents the effort that the controller should spend if there is no deviation at the time index k. This effort also takes into account of future disturbances encoded in Λ_k to Λ_N.

Proof. First, we will prove by backward induction that $J_k(I_k)$ as defined in Theorem 3.1.2 has the approximated reduced form:

$$J_k(I_k) = E_{x_k}\left[r_k + s_k'\delta x_k + \delta x_k' S_k \delta x_k | I_k\right]. \tag{3.90}$$

Indeed, we have for $k = N$:

$$J_N(I_N) = E_{x_N}\left[p_N + q_N'\delta x_N + \delta x_N' Q_N \delta x_N \middle| I_N\right], \tag{3.91}$$

$$S_N = Q_N, s_N = q_N, r_N = p_N. \tag{3.92}$$

By Theorem 3.1.2, we have the unconstrained minimization:

$$J_k(I_k) = \min_{u_k} E\left[\ell_k(x_k, u_k) + J_{k+1}(I_{k+1}) \middle| I_k, u_k\right]. \tag{3.93}$$

Assume:

$$J_{k+1}(I_{k+1}) = E_{x_{k+1}}\left[r_{k+1} + s_{k+1}'\delta x_{k+1} + \delta x_{k+1}' S_{k+1}\delta x_{k+1} \middle| I_{k+1}\right]. \tag{3.94}$$

Then:

$$J_k(I_k) = \min_{u_k} E\left[\ell_k(x_k, u_k) + r_{k+1} + s_{k+1}'\delta x_{k+1} + \delta x_{k+1}' S_{k+1}\delta x_{k+1} \middle| I_k, u_k\right] \tag{3.95}$$

$$= \min_{u_k} E\left[\ell_k(x_k, u_k) + E\left[r_{k+1} + s_{k+1}'\delta x_{k+1} + \delta x_{k+1}' S_{k+1}\delta x_{k+1} \middle| x_k, u_k\right] \middle| I_k, u_k\right]. \tag{3.96}$$

Expanding all sub terms with $\delta x_{k+1} = A_k \delta x_k + B_k \delta u_k + C_k w_k$:

$$\ell_k(x_k, u_k) = p_k + q_k' \delta x_k + \delta x_k' Q_k \delta x_k + t_k' \delta u_k + \delta u_k' T_k \delta u_k,$$
$$E\left[\delta x_{k+1}' S_{k+1} \delta x_{k+1} | x_k, u_k\right] = \delta x_k' A_k' S_{k+1} A_k \delta x_k + \delta u_k' B_k' S_{k+1} B_k \delta u_k + tr(C_k' S_{k+1} C_k \Omega_k^w)$$
$$+ \delta x_k' A_k' (S_{k+1} + S_{k+1}') B_k \delta u_k,$$
$$E\left[s_{k+1}' \delta x_{k+1} | x_k, u_k\right] = s_{k+1}' A_k \delta x_k + s_{k+1}' B_k \delta u_k.$$

Grouping all these terms in the form of:

$$J_k(I_k) = \min_{u_k} E\left[\delta x_k' S_k^- \delta x_k + s_k^{-\prime} \delta x_k + r_k^- + \delta u_k' H_k \delta u_k + (g_k + G_k \delta x_k)' \delta u_k \middle| I_k, u_k\right],$$

where

$$S_k^- = Q_k + A_k' S_{k+1} A_k, \tag{3.97}$$
$$s_k^- = (q_k' + s_{k+1}' A_k)', \tag{3.98}$$
$$r_k^- = r_{k+1} + p_k + tr(C_k' S_{k+1} C_k \Omega_k^w), \tag{3.99}$$
$$H_k = T_k + B_k' S_{k+1} B_k, \tag{3.100}$$
$$g_k = (t_k' + s_{k+1}' B_k)', \tag{3.101}$$
$$G_k = (A_k'(S_{k+1} + S_{k+1}') B_k)'. \tag{3.102}$$

Thus,

$$J_k(I_k) = E\left[\delta x_k' S_k^- \delta x_k + s_k^{-\prime} \delta x_k + r_k^- | I_k\right] + \min_{u_k} E\left[\delta u_k' H_k \delta u_k + (g_k + G_k \delta x_k)' \delta u_k \middle| I_k, u_k\right]. \tag{3.103}$$

Looking at related terms of u_k:

$$E\left[\delta u_k' H_k \delta u_k + (g_k + G_k \delta x_k)' \delta u_k \middle| I_k, u_k\right] = \delta u_k' H_k \delta u_k + (g_k + G_k \delta \hat{x}_k)' \delta u_k,$$

where $\delta \hat{x}_k = E[x_k | I_k] - \bar{x}_k = \hat{x}_k - \bar{x}_k$. Minimizing over δu_k without constraints, we

have:

$$(H_k + H_k')\delta u_k + g_k + G_k \delta \hat{x}_k = 0 \tag{3.104}$$

$$\delta u_k = -(H_k + H_k')^{-1}(g_k + G_k \delta \hat{x}_k) \tag{3.105}$$

$$\delta u_k = l_k + L_k \delta \hat{x}_k, \tag{3.106}$$

where

$$l_k = -(H_k + H_k')^{-1} g_k, \tag{3.107}$$

$$L_k = -(H_k + H_k')^{-1} G_k. \tag{3.108}$$

Hence:

$$\begin{aligned}
\delta u_k' H_k \delta u_k &= l_k' H_k l_k + \delta \hat{x}_k' L_k' H_k L_k \delta \hat{x}_k + l_k'(H_k + H_k') L_k \delta \hat{x}_k \\
g_k{}' \delta u_k &= g_k{}' l_k + g_k{}' L_k \delta \hat{x}_k \\
\delta x_k' G_k{}' \delta u_k &= l_k' G_k \delta x_k + \delta x_k' G_k{}' L_k \delta \hat{x}_k.
\end{aligned}$$

Substitute back into Eq. 3.103, we have $J_k(I_k) = E_{x_k}\left[r_k + s_k' \delta x_k + \delta x_k' S_k \delta x_k \middle| I_k\right]$ where:

$$\begin{aligned}
P_k &= L_k' H_k L_k + G_k{}' L_k, \\
S_k &= S_k^- + P_k = Q_k + A_k' S_{k+1} A_k + P_k, \\
s_k &= s_k^- + (l_k' G_k)' + (l_k'(H_k + H_k') L_k + g_k{}' L_k)' \\
&= (q_k' + s_{k+1}{}' A_k + l_k' G_k + l_k'(H_k + H_k') L_k + g_k{}' L_k)', \\
r_k &= r_k^- + l_k' H_k l_k + g_k{}' l_k - tr(P_k \Lambda_k), \\
&= r_{k+1} + p_k + tr(C_k' S_{k+1} C_k \Omega_k^w) + l_k' H_k l_k + g_k{}' l_k - tr(P_k \Lambda_k).
\end{aligned}$$

In the above mathematical manipulation, we use the following properties:

- $tr(A\Lambda_k) = tr(AVar[x_k|I_k]) = tr(AVar[\delta x_k|I_k]) = E[\delta x_k' A \delta x_k|I_k] - \delta \hat{x}_k' A \delta \hat{x}_k$,

- Assuming that H_k is invertible; in fact, with the objective functions of our models in Chapter 2, H_k is symmetric positive definite,

- In Eq. 3.103, we ignore the term r_k, which involves future conditional covariances in the term r_{k+1}. Under the *CE assumption*, we assume that future disturbances do not affect the choice of a control input at the current time. Due to this reason, a designed control law is CE-type locally-feedback.

Thus, we have proved Eq. 3.90. Furthermore, during this proof, from the Eq. 3.106, $\delta u_k = l_k + L_k \delta \hat{x}_k$, it infers that $u_k = \bar{u}_k + l_k + L_k(\hat{x}_k - \bar{x}_k)$. This is a function of information state I_k since $\hat{x}_k$ depends on I_k. Note that, we can show that H_k, S_x matrices are symmetric in the next lemma, and thus, what we need to prove follows. □

Lemma 3.3.2. *With the objective functions given in Section 2.7 and Section 2.8, H_k and S_k are symmetric positive definite matrices.*

Proof. With objective function in Section 2.7 and Section 2.8, the weight matrices $Q(\mathcal{T}), R(t)$. Therefore, we can check that Q_k, T_k are symmetric positive definite matrices in Lemma 3.3.1. For $k = N$, $S_N = Q_N$ is a symmetric positive definite matrix.

Assume that S_{k+1} is a symmetric positive definite matrix, we will show that H_k and S_k are positive definite matrices. Indeed, from Eq. 3.78, $H_k = T_k + B_k' S_{k+1} B_k$ is a positive definite matrix as it is a sum of two positive definite matrices. From Eq. 3.108 and Eq. 3.84, we have $S_k = A_k' S_{k+1} A_k + L_k' H_k L_k - \frac{1}{2} G_k' H_k G_k$, which is again a symmetric positive definite. □

The advantage of the unconstrained problem is that there is a closed form solution for a sub-optimal control law. With state and control constraints, a similar closed form is not available unless we use another form of approximation [7]. However, the next theorem proposes a heuristic approach to handle state and control constraints to keep the closed-form solution.

Theorem 3.3.2. *Given the* ***constrained*** *optimization problem in Eq. 2.12 to Eq. 2.19, there are heuristic modifications of l_k, L_k, S_k in Theorem 3.3.1 to obtain a sub-optimal control law having the following form $u_k = \mu_k(I_k) = \bar{u}_k + l_k + L_k(\hat{x}_k - \bar{x}_k)$.*

Proof. Recall that from Theorem 3.1.2, we have:

$$J_k(I_k) = \min_{u_k \in \mathcal{U}_k(I_k)} E\left[\ell_k(x_k, u_k) + J_{k+1}(I_{k+1}) \middle| I_k, u_k\right]. \tag{3.109}$$

We assume that from the mean value $\hat{x}_k$ of the EKF estimate, there is a simple or at least heuristic way to determine the feasible set $\mathcal{U}_k(I_k)$ so that x_{k+1} will not violate the feasible state set $\mathcal{X}$.

During the backward DP pass in Theorem 3.3.1, after calculating l_k, L_k as shown in Eq. 3.107 and Eq. 3.108, we have to modify these quantities due to constrained minimization over $\mathcal{U}_k(I_k)$. Recall that l_k represents the refined open-loop component for the current nominal control inputs, and L_k represents the feedback gain matrix for any deviation from a nominal trajectory. When there is no deviation, that is $\hat{x}_k - \bar{x}_k = 0$, $\bar{u}_k + l_k$ must belong to $\mathcal{U}_k(I_k)$. If it happens otherwise, we can make the following changes:

- Changing l_k such that $\bar{u}_k + l_k$ in $\mathcal{U}_k(I_k)$,
- Replacing those rows of L_k with zero rows which correspond to components of $\bar{u}_k + l_k$ on the boundary of $\mathcal{U}_k(I_k)$,
- Replace S_k with $\frac{1}{2}(S_k + S_k')$ as S_k has to be symmetric positive definite for the procedure to work in the next iteration.

□

Unlike the linear quadratic Gaussian (LQG) [3], due to the nonlinearity of system dynamics, given a control policy, the future conditional covariances are not constant. Thus, the numerical value of $J_k(I_k)$ in Eq. 3.90 cannot be computed directly using this recursive formula. Therefore, the iteration stops when a nominal open-loop control sequence converges. After obtaining a new open-loop control sequence u_k^{new}

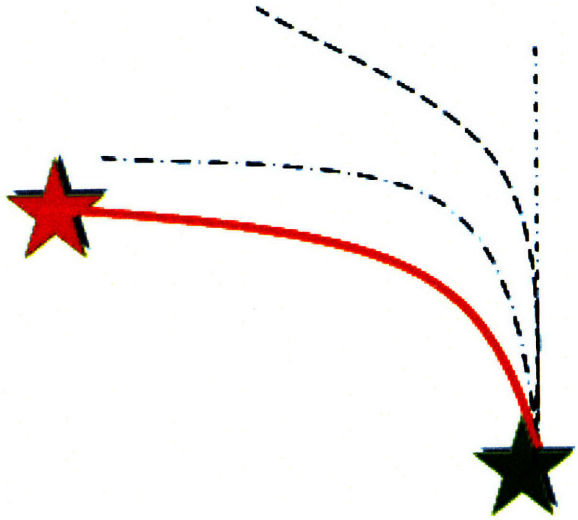


Figure 3-2: iLQG iteration.

by Theorems 3.3.1-3.3.2 and the EKF, a new candidate for the next nominal open-loop control sequence can be computed by $\bar{u}_k^{new} = \rho u_k^{new} + (1-\rho)\bar{u}_k$, where ρ is a real number within $[0, 1]$. Thus, the next iteration applies Theorem 3.3.1 again with the new nominal trajectory generated by $\bar{u}_k^{new}$. Figure 3-2 illustrates this procedure graphically. As we can see, the green star and the red star represent a starting location and a destination respectively. The dashed lines are nominal trajectories generated by nominal open-loop control inputs in first few iterations. The bold solid line represents a converged trajectory generated by converged open-loop control inputs. We note that the shape of a converged trajectory depends on the definition of an objective function and its weight matrices.

Algorithm 1 shows how the iLQG algorithm works. The algorithm takes the dynamics f, g, the formulas of h, l_k, a planned horizon N, and an initial distribution $N(x_0^-, \Lambda_0^-)$ as input parameters, and it returns $\bar{u}, l, L$ to establish a locally-feedback control policy. As we can see, one loop in the iLQG algorithm has three main steps: linearizing system dynamics and approximating cost function (Lemma 3.3.1), solving a local linear quadratic Gaussian problem (Theorem 3.3.1 and Theorem 3.3.2), and updating a new nominal control policy.

3.4 POBRM: offline phase

In this section, we start focusing on the context of planning and controlling a robot. In particular, we study how to use a feature-based map to build a belief graph with associated transfer functions in the offline phase. That is, we construct a belief graph by probabilistically sampling points around the features that potentially provide the robot with valuable information. Each edge of the belief graph stores two transfer functions to predict the cost and the conditional covariance matrix of a final state estimate if the robot follows this edge. The constructed information in the offline phase is used for multiple queries efficiently in the online phase.

Algorithm 1 iLQG algorithm

procedure iLQG($f, g, h, \ell, N, x_0^-, \Lambda_0^-$)
  Receive an initial measurement y_0
  Perform the EKF update to get $N(x_0, \Lambda_0)$ (Theorem 3.2.1)
  Initialize $\bar{u} \leftarrow 0, l \leftarrow 0, L \leftarrow 0$
  while !converged **do**
    Compute $\bar{x}$ (Eq. 3.60)
    Compute $A_k, B_k, C_k, Q_N, q_N, p_N, Q_k, q_k, T_k, t_k, p_k$ (Eq. 3.61 - 3.74)
    $S_N \leftarrow Q_N, s_N \leftarrow q_N, r_N \leftarrow p_N$
    for $k \leftarrow N-1, ..., 0$ **do**
      Compute $\mathcal{U}_k(I_k)$ based on $\bar{x}_k$
      Compute $H_k, l_k, L_k, P_k, g_k, G_k, S_k, s_k, r_k$ (Eq. 3.81 - 3.89)
      if $\bar{u_k} + l_k \notin \mathcal{U}_k(I_k)$ **then**
        Fix l_k
        Replace corresponding rows of L_k with zero rows
        $S_k \leftarrow \frac{1}{2}(S_k + S_k')$
      end if
    end for
    for $k \leftarrow 0, ..., N-1$ **do**
      Simulate a new control law $u_k^{new} \leftarrow \bar{u}_k + l_k + L_k(\hat{x}_k - \bar{x}_k)$
    end for
    Update $\bar{u} \leftarrow \rho u^{new} + (1-\rho)\bar{u}$
  end while
  return $\bar{u}, l, L$
end procedure

3.4.1 Building a belief graph

Definition 3.4.1. *(Visibility graph) Let C denote the space of all robot poses, C_{free} denote the set of all collision-free poses, and C_{obst} denote the space of all robot poses resulting in collision with obstacles. We have $C = C_{free} \cup C_{obst}$. The approximation visibility graph of C_{free} can be constructed by sampling poses in C_{free} as vertices and connecting straight lines between vertices lying entirely in C_{free} as edges.*

In fully observable problems, the Probabilistic Roadmap algorithm [10] computes a trajectory between two locations by searching induced visibility graphs. As we are interested in partially observable problems, induced visibility graphs cannot be constructed since poses of a robot are not directly accessible. Nevertheless, a similar approach, called the belief graph, is presented in [22], [8] for partially observable cases.

Since a robot does not know its state directly, it only has access to EKF state estimates $b_k = N(\hat{x}_k, \Lambda_k)$. One the one hand, if the norm of a covariance matrix Λ_k is large, a robot is not confident about its actual pose. On the other hand, if the norm of a covariance matrix Λ_k is small, a robot has a high confidence in its pose estimate. A naive approach as suggested by the visibility graph concept is to sample directly in the belief space in the form $b_k = N(\hat{x}_k, \Lambda_k)$. However, the probability to sample all reachable beliefs, including means and covariances, are almost zero as the belief space is exponentially large. Instead, an induced belief graph can be constructed as follows. Reachable means x_k^s of beliefs $b_k = N(x_k^s, \Lambda_k^s)$ can be sampled as done in visibility graphs to be vertices in a belief graph. Edges are straight lines in the free space C_{free}, connecting pairs of vertices. The reachable covariance Λ_k^s at a vertex depends on the covariance Λ_j^s of a departing vertex along a connecting edge $b_j b_k$. The reachable covariances along a path can be propagated using the EKF from a starting vertex (Figure 3-4).

Definition 3.4.2. *(Belief graph) Let C denote the space of all robot poses, C_{free} denote the set of all collision-free poses, and C_{obst} denote the space of all robot poses resulting in collision with obstacles. We have $C = C_{free} \cup C_{obst}$. The approximation belief graph of C_{free} can be constructed by sampling mean poses in C_{free} as vertices and connecting*

Algorithm 2 Constructing a belief graph algorithm

```
1: procedure BUILDGRAPH(map)
2:     G.V ← ∅
3:     G.E ← ∅
4:     for all feature m do
5:         for i ← 1, 2, ..., numberofsamples do
6:             Sample x^s ∈ C_free
7:             Add b = (x^s, Λ = ∅) to set G.V
8:         end for
9:     end for
10:    for all vertex b_i do
11:        for all vertex b_j (i ≠ j) do
12:            if x_i^s x_j^s ∈ C_free then
13:                Add edge b_i b_j to set G.E
14:            end if
15:        end for
16:    end for
17:    return G
18: end procedure
```

straight lines between vertices lying entirely in $\mathcal{C}_{free}$ as edges. For each vertex in a belief graph, the reaching covariance is computed online for each control sequence to reach that vertex.

There are different strategies to sample vertices of a belief graph. As a simple approach, the mean components of beliefs can be sampled randomly throughout the free configuration space $\mathcal{C}_{free}$ like in the PRM algorithm. However, in a feature-based map, the area around features will provide rich information for a robot. Therefore, sampling vertices randomly around features allows for good planning with less vertices. In this research, this strategy is used. Alternatively, as presented in [8], a more complicated sampling strategy based on the information Theorem around features can be used.

Figure 3-3 depicts a strategy to sample means of beliefs in a feature-based map. The red dots are recognizable features, and the yellow dots are sampled means nearby features. Connecting these vertices by straight lines in the free space $\mathcal{C}_{free}$, we have a belief graph in Figure 3-4. As we can see, each vertex has two components x_k^s and Λ_k^s. Moreover, each edge is associated with two transfer function ξ and $\tilde{J}$, which will

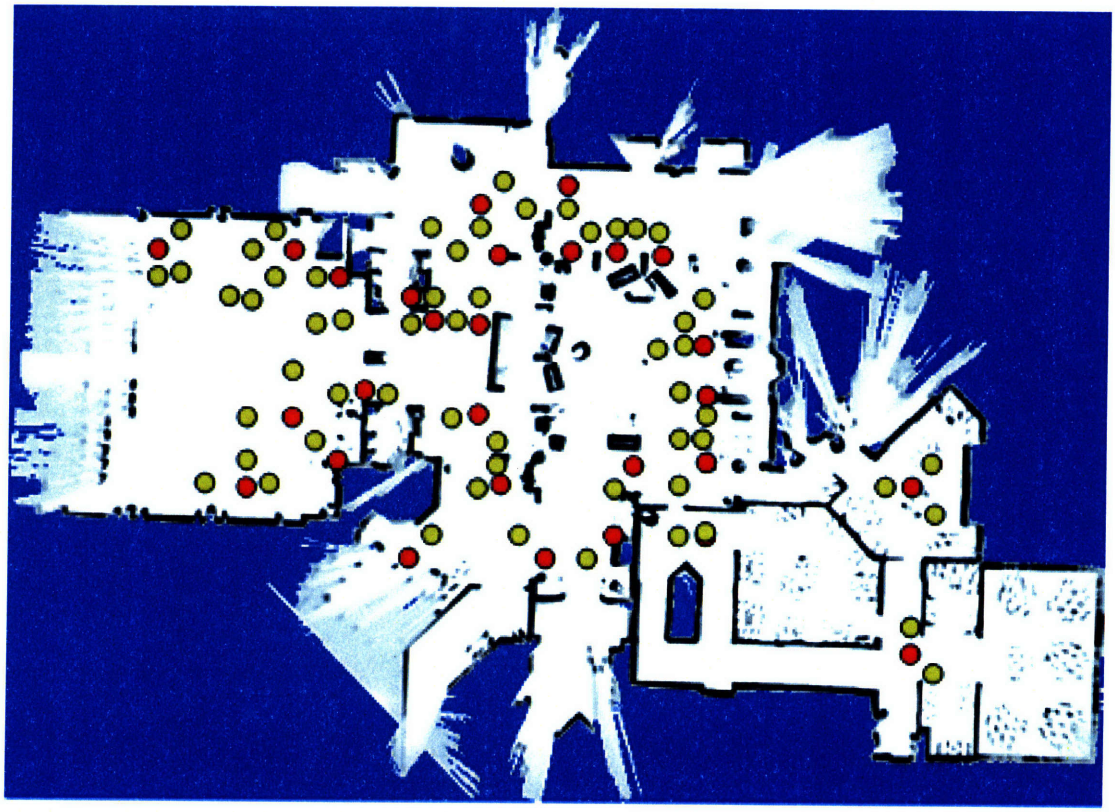


Figure 3-3: Sampling a feature-based map.

be constructed later.

Algorithm 2 summarizes the procedure to construct a belief graph from a feature-based map. The algorithm takes a feature-based map, which consists of a list of feature coordinates, as an input parameter. A graph $\mathcal{G}$ consisting of a set of vertices and a set of edges is returned.

3.4.2 Computing a locally-feedback control law for an edge

With a constructed belief graph $\mathcal{G}$, the offline phase of the POBRM algorithm starts building a database for the graph. First, a horizon N to traverse each edge is determined heuristically by computing a number of time steps to traverse the edge's length with a nominal velocity v_0. Second, a locally-feedback control law to traverse an edge of the belief graph can be computed by solving the following subproblem using the iLQG algorithm.

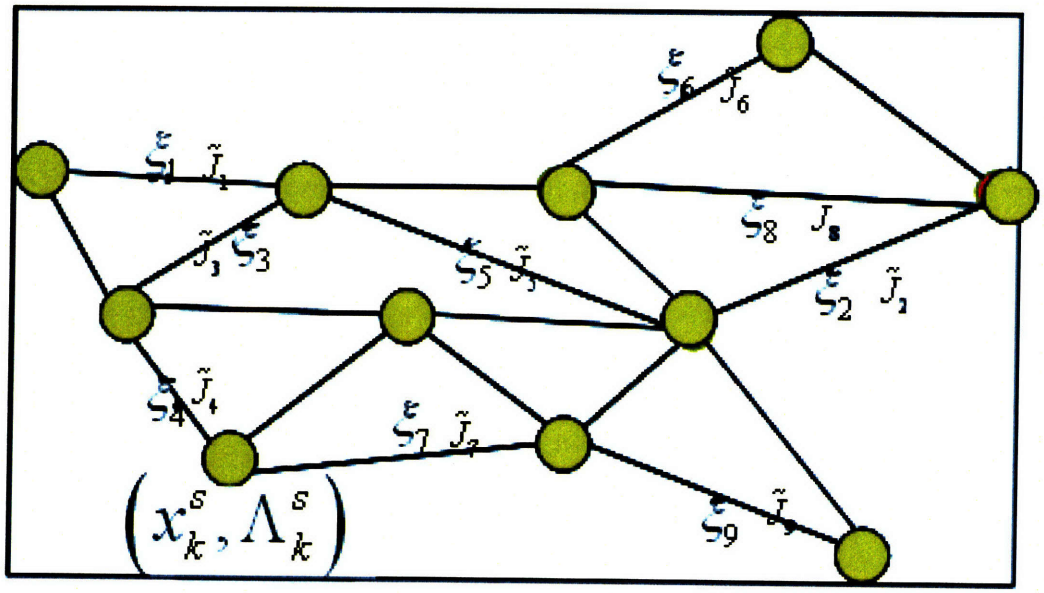


Figure 3-4: Belief graph.

Definition 3.4.3. *(CE locally-feedback control law to traverse an edge) Let* $b_i = N(x_i^s, \Lambda_i^s)$ *and* $b_j = N(x_j^s, \Lambda_j^s)$ *be two beliefs, which are vertices in a belief graph. A locally-feedback control law of CE type* π^{ij} *to traverse along the edge connecting* x_i^s *and* x_j^s *is defined in the following optimization problem:*

$$N = N^{ij} = \frac{distance(x_i^s, x_j^s)}{v_0}, \tag{3.110}$$

$$h(x_N) = (x_N - x_j^s)' Q_N (x_N - x_j^s), \quad Q_N \succ 0, \tag{3.111}$$

$$\ell_k(x_k, u_k) = u_k' R_k u_k, \quad R_k \succ 0, \tag{3.112}$$

$$\min_{\Pi} E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k) \middle| I_0\right] \tag{3.113}$$

subject to:

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k\sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{3.114}$$

$$y_k = g(x_k) + \vartheta_k, \qquad k = 0, 1, ..., N \tag{3.115}$$

$$I_k = [y_0, y_1, ..., y_k, u_0, u_1, ..., u_{k-1}], \tag{3.116}$$

$$u_k = \mu_k(I_k), \tag{3.117}$$

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{3.118}$$

$$x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, y_k \in \mathbb{R}^{n_y}, w_k \in \mathbb{R}^{n_w}, \vartheta_k \in \mathbb{R}^{n_\vartheta}, \tag{3.119}$$

$$w_k \sim N(0, \Omega^w), \vartheta_k \sim N(0, \Omega^\vartheta), x_0 \sim N(x_i^s, \Lambda_0^s), I_0 = y_0. \tag{3.120}$$

This subproblem is solved by using the iLQG algorithm in Theorem 3.3.1.

3.4.3 Constructing covariance transfer functions

Then, under the mild assumption of environments, the final conditional covariance Λ_j^s can be predicted from a transfer function having a starting covariance Λ_i^s as an input parameter.

Assumption 3.4.1. *(Mild environment) We assume that disturbances in an operating environment are unbiased and not too adverse to distort the motion of a robot severely. Thus, under the same sequence of control inputs, a robot will receive similar amount of information from received measurements, which is known as a priori.*

Recall that the covariance Λ_j^s can be found by the EKF dynamics for every new query with different Λ_i^s. However, the simulation of the EKF dynamics involves an inverse operation for every incoming measurement. Thus, this is computationally intense for high-dimensional systems, and therefore it is desirable to have a more efficient approach to estimate Λ_j^s for all new queries. This demand is illustrated in Figure 3-5. Theorem 3.4.1 proposes a solution to this demand.

Theorem 3.4.1. *(Covariance transfer function) Given a control law π^{ij} to traverse an edge $b_i b_j$, there is a linear operator ξ^{ij} such that the conditional covariance of a belief b_j can be predicted as*

$$[\Psi] = [\xi^{ij}] \begin{bmatrix} \Lambda_i^s \\ I \end{bmatrix}, \tag{3.121}$$

$$\Lambda_j^s = [\Psi_{1,1}] [\Psi_{2,1}]^{-1}. \tag{3.122}$$

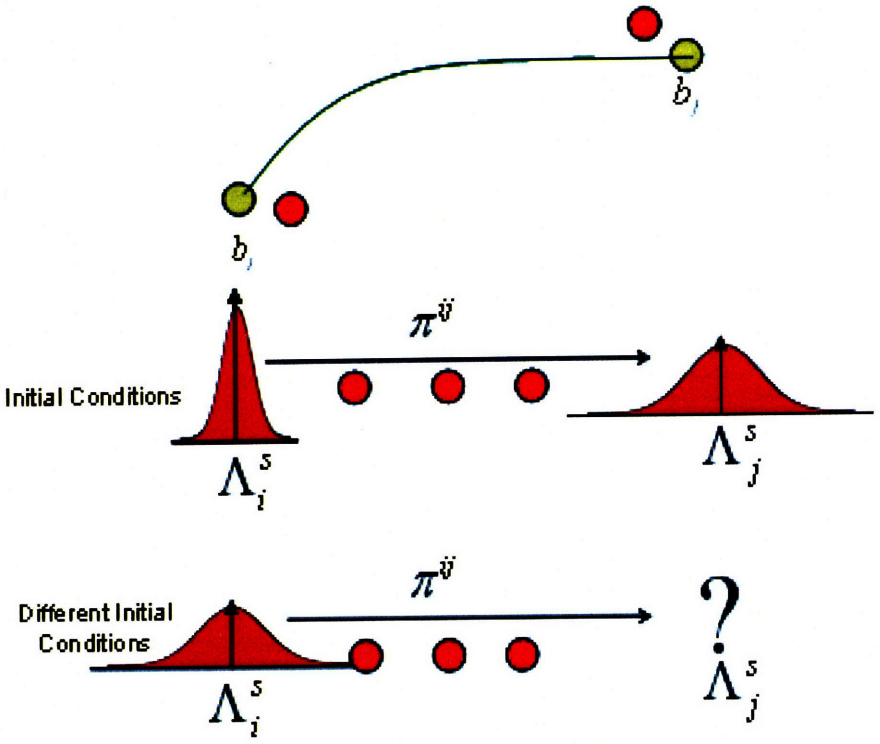


Figure 3-5: New query of the final covariance.

Proof. Using the control law π^{ij} to simulate a trajectory once, the controller provides a control consequence $\{u_0, u_1, ..., u_{N-1}\}$, and a robot receives a measurement sequence $\{y_1, y_2, ..., y_{N-1}, y_N\}$. From Theorem 3.2.2, let $\Lambda_i^s = \Lambda_i^s I^{-1}$, we can construct matrices $\xi_1^-, \xi_1, \xi_2^-, ..., \xi_N^-, \xi_N$ such that:

$$[\Psi] = \begin{bmatrix} \mathcal{B}_N \\ \mathcal{C}_N \end{bmatrix} = [\xi_N] \left[\xi_N^-\right] ... [\xi_1] \left[\xi_1^-\right] \begin{bmatrix} \Lambda_i^s \\ I \end{bmatrix}. \tag{3.123}$$

So:

$$\xi^{ij} = [\xi_N] \left[\xi_N^-\right] ... [\xi_1] \left[\xi_1^-\right], \tag{3.124}$$

$$\Lambda_j^s = [\Psi_{1,1}] [\Psi_{2,1}]^{-1}. \tag{3.125}$$

Under mild environments, we expect that generated control inputs and received measurements would not be significantly different in several queries. Thus, the matrix

$\xi^{i,j}$, which is defined as a covariance transfer function, is *computed once* using simulation of the control law π^{ij} and associated with the edge $b_i b_j$. Given the matrix $\xi^{i,j}$, this theorem implies that a conditional covariance of an estimate after executing a series of control inputs and receiving a series of measurements can be computed by linear operations and only one matrix inversion operation from an initial conditional covariance. This significantly improves the computational speed of the EKF. Therefore, we can use this transfer function to efficiently predict a covariance at vertex b_j given a covariance at vertex b_i in future queries. Figure 3-6 illustrates graphically the proof. □

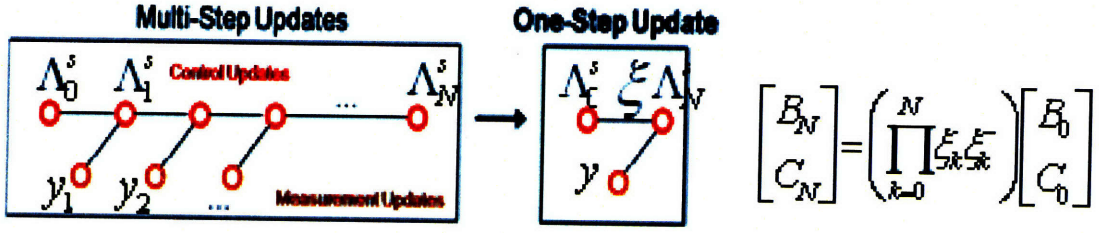


Figure 3-6: Mechanism of covariance transfer functions.

3.4.4 Constructing cost transfer functions

Finally, the cost to traverse an edge $b_i b_j$ using a control law π^{ij} varies with respect to a departing covariance Λ_i^s. Similar to covariance transfer functions, we need a method to estimate the cost-to-go efficiently for multiple queries. Using an approximated cost structure, we can parameterize this objective value as a function of Λ_i^s so that future queries can estimate quickly the corresponding objective values.

From the proof of Theorem 3.3.1, Eq. 3.90 suggests that the objective function value can be approximated as

$$J_0(I_0) = E_{x_0}\left[r_0 + s_0'\delta x_0 + \delta x_0' S_0 \delta x_0 | I_0\right] \tag{3.126}$$

$$\approx \widetilde{r}_0 + (\widetilde{s}_0)'\delta\hat{x}_0 + \delta\hat{x}_0'\widetilde{S}_0\delta\hat{x}_0 + tr(\widetilde{P}\Lambda_i^s) \tag{3.127}$$

$$= \widetilde{r}_0 + tr(\widetilde{P}\Lambda_i^s). \tag{3.128}$$

The first approximation is due to $E_{x_0}\left[\delta x_0' S_0 \delta x_0 | I_0\right] = \delta\hat{x}_0' S_0 \delta\hat{x}_0 + tr(S_0\Lambda_i^s)$, and from Eq. 3.86, r_0 contains terms relating to future covariances Λ_ks. Therefore, we approximate the contribution of future covariances as an uncertainty weight matrix $\widetilde{P}$, and approximate r_0 as $\widetilde{r}_0$. The last equality is due to $\delta\hat{x}_0 = \hat{x}_0 - x_i^s = x_i^s - x_i^s = 0$, which states that the mean of an initial robot's pose is a sampled pose at a vertex b_i. Although the exact value of $\widetilde{r}_0$ and $\widetilde{P}$ cannot be computed, we will show in Theorem 3.4.2 to learn these values using stochastic iterative algorithms [4].

Normally, a sub-optimal objective value to traverse an edge resulted from an optimization problem with an objective function including a terminating cost component is approximated. However, we are also interested in objective functions without a terminating cost component when incoming vertices are just transiting waypoints in a long trajectory. Thus, we differentiate two types of cost transfer functions and provide a mechanism to iteratively find the approximation structure.

Definition 3.4.4. *The type one (type-1) cost transfer function of an edge approximates the cost-to-go of a control law* ***without*** *the final terminating cost:*

$$J_\pi^1(I_0) = E\left[\sum_{k=0}^{N-1} \ell_k(x_k, u_k)\Big| I_0\right]. \tag{3.129}$$

The type two (type-2) cost transfer function of an edge approximate the cost-to-go of a control law ***with*** *the final terminating cost:*

$$J_\pi^2(I_0) = E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k)\Big| I_0\right]. \tag{3.130}$$

Theorem 3.4.2. *(Cost transfer function) The approximated cost function $\widetilde{J}^{ij}$ having the form $\widetilde{r}_0 + tr(\widetilde{P}\Lambda_i^s)$ estimates the cost-to-go along the edge $b_i b_j$ with a control law π^{ij}. Starting with arbitrary values of $\widetilde{r}_0$ and $\widetilde{P}$, the following iterative procedure provides a numerical solution to the approximated values of $\widetilde{r}_0$ and $\widetilde{P}$:*

- *Initialize a random initial covariance Λ_i^s,*
- *Computing the current approximated value J_0^c as $\widetilde{r}_0 + tr(\widetilde{P}\Lambda_i^s)$,*

- *Simulate type-1 or type-2 value J_0^s of J_π according to the fixed control π^{ij},*
- *Updating $\widetilde{r}_0$ and $\widetilde{P}$ as*

$$\widetilde{r}_0^{new} = \widetilde{r}_0 - \gamma(J_0^c - J_0^s), \tag{3.131}$$

$$\widetilde{P}^{new} = \widetilde{P} - \gamma(J_0^c - J_0^s)\Lambda_0^s. \tag{3.132}$$

The value of γ approaches to 0 as the number of iterative steps increases.

In the above equations, the term $\widetilde{r}_0$ approximates the effort to reach the vertex b_j when there is no uncertainty at the vertex b_i. In addition, the weight matrix $\widetilde{P}$ approximates the contribution of uncertainty at the vertex b_i and future induced uncertainty in the cost-to-go.

Proof. Given the current values of $\widetilde{r}_0$ and $\widetilde{P}$, finding the next approximated values of these variables is equivalent to solving the following optimization problem:

$$\min_{\widetilde{r}_0, \widetilde{P}} \frac{1}{2}\left(\widetilde{r}_0 + tr(\widetilde{P}\Lambda_i^s) - J_0^s\right)^2. \tag{3.133}$$

Denote:

$$L(\widetilde{r}_0, \widetilde{P}) = \frac{1}{2}\left(\widetilde{r}_0 + tr(\widetilde{P}\Lambda_i^s) - J_0^s\right)^2 \tag{3.134}$$

$$= \frac{1}{2}\left(\widetilde{r}_0 + E\left[(x_0 - \hat{x}_0)'\widetilde{P}(x_0 - \hat{x}_0)\middle| I_0\right] - J_0^s\right)^2. \tag{3.135}$$

Using the gradient method, the next numerical solution is:

$$\widetilde{r}_0^{new} = \widetilde{r}_0 - \gamma\nabla_{\widetilde{r}_0} L(\widetilde{r}_0, \widetilde{P}) \tag{3.136}$$

$$= \widetilde{r}_0 - \gamma(\widetilde{r}_0 + tr(\widetilde{P}\Lambda_i^s) - J_0^s) \tag{3.137}$$

$$= \widetilde{r}_0 - \gamma(J_0^c - J_0^s), \tag{3.138}$$

$$\widetilde{P}^{new} = \widetilde{P} - \gamma\nabla_{\widetilde{P}} L(\widetilde{r}_0, \widetilde{P}) \tag{3.139}$$

$$= \widetilde{P} - \gamma(\widetilde{r}_0 + tr(\widetilde{P}\Lambda_i^s) - J_0^s)(E\left[(x_0 - \hat{x}_0)(x_0 - \hat{x}_0)'\middle| I_0\right]) \tag{3.140}$$

$$= \widetilde{P} - \gamma(J_0^c - J_0^s)\Lambda_i^s. \tag{3.141}$$

In the above equations, γ is a step size in the gradient method. To ensure convergence, it is well known that γ should approach 0 when the number of iteration increases [4]. The convergence criteria for this procedure is when changes in norms of $\widetilde{r}_0$ and $\widetilde{P}$ are below some thresholds. □

Once the value of $\widetilde{r}_0$ and $\widetilde{P}$ for an edge are determined by Theorem 3.4.2, we have established the cost transfer function $\widetilde{J}$ with an initial covariance matrix as an input parameter for that edge. Thus, for any future query to traverse the edge, the associated cost transfer function can predict rapidly the cost-to-go by plugging an initial covariance into the cost transfer function.

The offline phase of the POBRM algorithm is summarized in Algorithm 3. The algorithm requires a feature-based map and returns a graph $\mathcal{G}$ having vertex set, edge set and associated transfer functions. Overall, there are two main parts in the algorithm, namely sampling a belief graph (Algorithm 2), building covariance transfer functions and cost transfer functions (Theorems 3.4.1 - 3.4.2).

Algorithm 3 Offline phase of the POBRM

procedure OFFLINEPHASE(map)
  $\mathcal{G}$= buildGraph(map) (Algorithm 2)
  for all edge $b_ib_j \in \mathcal{G}.E$ **do**
    $N \leftarrow distance(x_i^s, x_j^s)/v_0$
    $h(x_N) = (x_N - x_j^s)'Q_N(x_N - x_j^s)$
    $\ell_k(x_k, u_k) = u_k'R_ku_k$
    $\pi^{ij} = iLQG(f, g, h, \ell, N, x_i^s, \Lambda_i^s)$ (Algorithm 1)
    Simulate π^{ij} to compute $b_ib_j.\xi \leftarrow [\xi_N]\left[\xi_N^-\right] \ldots [\xi_1]\left[\xi_1^-\right]$ (Theorem 3.4.1)
    Learn $b_ib_j.\widetilde{J}$ (Theorem 3.4.2)
  end for
  return $\mathcal{G}$
end procedure

3.5 POBRM: online phase

In this section, we use a belief graph built in the offline phase to perform the Dijkstra's search to plan a sub-optimal trajectory in terms of waypoints. Then, a locally-

feedback control law to follow the sub-optimal trajectory is obtained by the iLQG algorithm.

3.5.1 Sub-optimal trajectory

Given a belief graph $\mathcal{G}$ with covariance transfer functions and cost transfer functions associated with its edges, the mission is to plan a trajectory from an initial belief $N(x_0^-, \Lambda_0^-)$ to a destination with the final state x_G.

First, the initial starting location and the destination are added into the belief graph. In addition, we construct two additional nearest edges, *from* the initial location and *to* the destination, and their corresponding transfer functions. The edge linking with the initial location has a cost transfer function of type-1, and the edge linking with the destination has a cost function of type-2. Moreover, we also construct a direct edge from the initial location to the destination with a type-2 cost transfer function.

Second, from the initial belief, covariances at other vertices can be computed efficiently by propagating Λ_0^- using covariance transfer functions. Then, the approximated cost to traverse an edge is computed by plugging a covariance at a departing vertex into the associated cost transfer function of that edge. We consider these cost values as edge weights, and therefore the Dijkstra's search can be applied to find a trajectory with the smallest cost-to-go. Normally, a large covariance at a departing vertex results in a large edge weight. Therefore, this Dijkstra's search provides us with a trade-off between keeping the covariance of a robot state estimate small and keeping the energy to find a destination small. In other words, a balance between exploration and exploitation is achieved in the online phase of the POBRM algorithm.

3.5.2 Sub-optimal controller

Suppose that the Dijkstra's search provides a trajectory α with ordered waypoints $\{b_{\alpha_1}, b_{\alpha_2}, ..., b_{\alpha_m}\}$, and the corresponding horizons to reach these vertices from its preceding vertices are $\{N_{\alpha_1}, N_{\alpha_2}, ..., N_{\alpha_m}, N_{\alpha_{m+1}}\}$. In particular, there are N_{α_1} time

steps to traverse from the initial location to a vertex b_{α_1}, and there are $N_{\alpha_{m+1}}$ time steps to traverse from a vertex b_{α_m} to the final destination. Let N be a total number of time steps to follow the trajectory, and k_j is the time index to visit a vertex b_{α_j}. We have:

$$N = N_{\alpha_1} + N_{\alpha_2} + ... + N_{\alpha_m} + N_{\alpha_{m+1}}, \tag{3.142}$$

$$k_j = N_{\alpha_1} + N_{\alpha_2} + ... + N_{\alpha_j}. \tag{3.143}$$

The corresponding objective function to go from the initial location to a destination location via these waypoints in an induce optimization problem are defined as

$$h(x_N) = (x_N - x_G)' Q_N (x_N - x_G), \quad Q_N \succ 0, \tag{3.144}$$

$$\ell_k(x_k, u_k) = u_k' R_k u_k, \quad R_k \succ 0, \quad k \notin \{k_1, k_2, ..., k_m\}, \tag{3.145}$$

$$\ell_{k_j}(x_{k_j}, u_{k_j}) = u_{k_j}' R_{k_j} u_{k_j} + (x_{k_j} - x_{\alpha_j}^s)' Q_{k_j} (x_{k_j} - x_{\alpha_j}^s), \quad R_{k_j} \succ 0, Q_{k_j} \succ 0, \tag{3.146}$$

$$\min_{\Pi} E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k) \middle| I_0 \right] \tag{3.147}$$

subject to:

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k \sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{3.148}$$

$$y_k = g(x_k) + \vartheta_k, \qquad k = 0, 1, ..., N \tag{3.149}$$

$$I_k = [y_0, y_1, ..., y_k, u_0, u_1, ..., u_{k-1}], \tag{3.150}$$

$$u_k = \mu_k(I_k), \tag{3.151}$$

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{3.152}$$

$$x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, y_k \in \mathbb{R}^{n_y}, w_k \in \mathbb{R}^{n_w}, \vartheta_k \in \mathbb{R}^{n_\vartheta}, \tag{3.153}$$

$$w_k \sim N(0, \Omega^w), \vartheta_k \sim N(0, \Omega^\vartheta), x_0 \sim N(x_0^-, \Lambda_0^-), I_0 = y_0. \tag{3.154}$$

Again, the above optimization can be solved sup-optimally by the iLQG algorithm to obtain a locally-feedback control law π.

Figure 3-7 shows the operations of the online phase. In particular, two new vertices

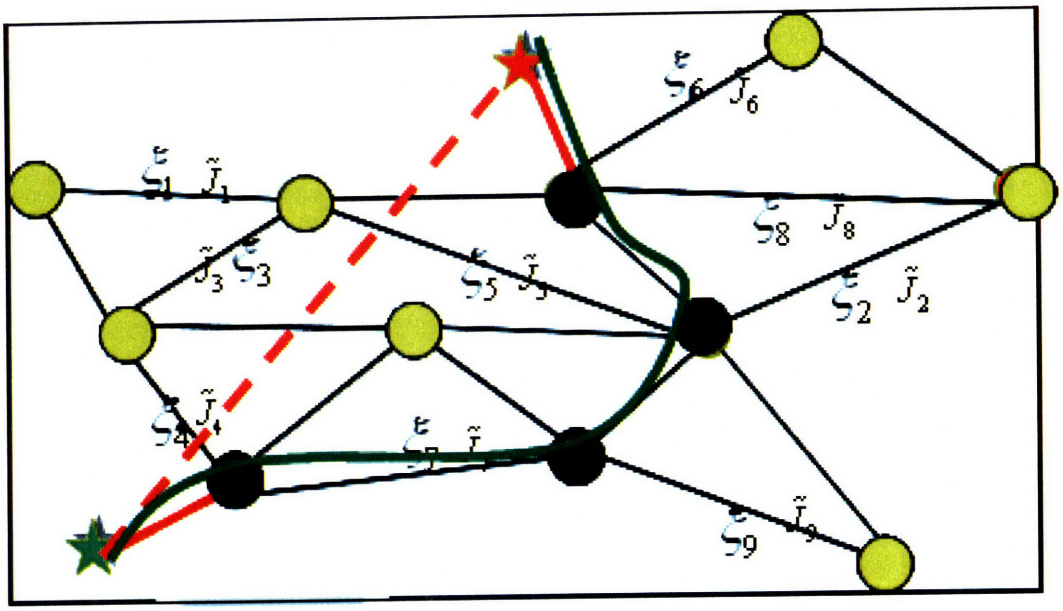


Figure 3-7: Illustration of the online phase.

are the starting green star and the final red star. Three additional edges are depicted in red lines. After the Dijkstra's search, the found waypoints are in black dots. Finally, the iLQG refines a sub-optimal trajectory in the green curve and returns a locally-feedback control law π as well.

The overall summary of the POBRM online phase is presented in Algorithm 4. Input parameters are a belief graph $\mathcal{G}$, an initial belief $N(x_0^-, \Lambda_0^-)$, and a destination state x_G. The online phase returns a sub-optimal trajectory x and control law π.

3.6 Sources of error

First, since we actually solve the discrete-time formulation, the obtained solution approximates to the solution of the continuous counterpart. If the time step Δ is small, the error between these two solutions may be negligible. It is likely that when the time step Δ approaches 0, a discrete-time solution approaches a continuous-time solution. However, this phenomena does not always hold since the discretization may also lead to the problem of inconsistency.

Second, when constructing a belief graph, the more vertices are in the graph, the

Algorithm 4 Online phase of the POBRM

procedure ONLINEPHASE($\mathcal{G}$, $N(x_0^-, \Lambda_0^-)$,x_G)
  Receive an initial measurement y_0
  Perform the EKF update to get $N(x_0, \Lambda_0)$ (Theorem 3.2.1)
  $gz \leftarrow size(\mathcal{G}.V)$
  Add vertices $b_{gz+1} = (x_0, \Lambda_0)$, $b_{gz+2} = (x_G, \Lambda_G = \emptyset)$ to $\mathcal{G}.V$
  Add nearest type-1 edge $b_{gz+1}b_{j_1}$ to $\mathcal{G}.E$
  Add nearest type-2 edge $b_{j_2}b_{gz+2}$ to $\mathcal{G}.E$
  Add direct type-2 edge $b_{gz+1}b_{gz+2}$ to $\mathcal{G}.E$
  for all $b \in \mathcal{G}.V$ **do**
    $cost[b] \leftarrow \infty$
    $prev[b] \leftarrow -1$;
  end for
  $cost[b_{gz+1}] \leftarrow 0$
  $Queue \leftarrow \mathcal{G}.V$
  while $!empty(Queue)$ **do**
    $b \leftarrow popMin(Queue)$
    if $b! = b_{gz+2}$ **then**
      break
    end if
    for all neighbor v of b **do**
      $[\Psi] \leftarrow [bv.\xi] \begin{bmatrix} b.\Lambda \\ I \end{bmatrix}$
      $\Lambda \leftarrow [\Psi_{1,1}] [\Psi_{2,1}]^{-1}$
      $alt \leftarrow cost[b] + bv.\widetilde{J}(\Lambda)$
      if $alt < cost[v]$ **then**
        $cost[v] \leftarrow alt$
        $v.\Lambda \leftarrow \Lambda$
        $prev[v] \leftarrow b$
      end if
    end for
  end while
  Compute waypoints $wp \leftarrow traceBack(prev)$ ▷ Assume $wp = \{b_{\alpha_1}, ..., b_{\alpha_m}\}$
  $N \leftarrow N_{\alpha_1} + ... + N_{\alpha_{m+1}}$
  for $j \leftarrow 1, ..., m$ **do**
    $k_j \leftarrow N_{\alpha_1} + ... + N_{\alpha_j}$
  end for
  $h(x_N) = (x_N - x_G)' Q_N (x_N - x_G)$
  $\ell_k(x_k, u_k) = u_k' R_k u_k, \quad k \notin \{k_1, k_2, ..., k_m\}$
  $\ell_{k_j}(x_{k_j}, u_{k_j}) = u_{k_j}' R_{k_j} u_{k_j} + (x_{k_j} - x_{\alpha_j}^s)' Q_{k_j} (x_{k_j} - x_{\alpha_j}^s)$
  $\pi = iLQG(f, g, h, \ell, N, x_0, \Lambda_0)$
  Simulate the control law π to get a trajectory x
  return x, π
end procedure

better the solution is. Ideally, if sampled vertices are identical to the best trajectory, a returned solution will be identical to a globally optimal solution. However, it is impossible to sample the best trajectory, and sampled vertices scatter randomly in the free space of a map. Therefore, sampling vertices of a belief graph is a second source of error.

Third, the iLQG algorithm, which is used to find a sub-optimal controller along each edge, produces error when it stops the loop to look for a nominal trajectory. First, the algorithm stops the loop when some tolerance is met, so a converged nominal trajectory bears numerical error. Second, a nominal trajectory does not purposely provide information to increase the confidence of the robot state, while the best trajectory may detour via some features. This is, ofcourse, the original problem that we wish to solve.

Fourth, under a control law π^{ij} for traversing an edge, the corresponding covariance and cost transfer functions are other sources of error. We assumed that the environment is not so pathological to provide substantially different trajectories and observations occur in different runs of the same control law π^{ij}. However, these transfer functions will provide inaccurate predictions of resulting covariances and cost-to-go values if the noise in the environment is too dominant.

Fifth, in the final run of the iLQG to find a control law going through waypoints to reach a destination, the objective consists of additional terms to enforce a robot going through the waypoints. Thus, the cost-to-go value in the final iLQG problem will be different from the value of the original problem that is suggested in the Dijkstra's search. This discrepancy will also result in error between an obtained trajectory and a globally optimal trajectory.

3.7 Analysis of the POBRM algorithm

Let us assume that the first four types of error are negligible, we will address the remaining error to show that it is possible to bound the sub-optimal cost-to-go compared to the globally optimal cost-to-go. That is, we first assume that discretization

scheme is consistent, a belief graph is sampled well enough, and all transfer functions predict covariance and cost-to-go for an edge accurately. Therefore, the Dijkstra's search is able to pick up waypoints $\{b_{\alpha_1}, b_{\alpha_2}, ..., b_{\alpha_m}\}$ along a globally optimal trajectory α.

Recall that the original discrete-time optimization problem is

$$J^* = \min_{\Pi} E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k)\Big| I_0\right] \tag{3.155}$$

subject to:

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k\sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{3.156}$$

$$y_k = g(x_k, u_k) + \vartheta_k, \qquad k = 0, 1, ..., N \tag{3.157}$$

$$I_k = [y_0, y_1, ..., y_k, u_0, u_1, ..., u_{k-1}], \tag{3.158}$$

$$u_k = \mu_k(I_k), \tag{3.159}$$

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{3.160}$$

$$x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, y_k \in \mathbb{R}^{n_y}, w_k \in \mathbb{R}^{n_w}, \vartheta_k \in \mathbb{R}^{n_\vartheta}, \tag{3.161}$$

$$w_k \sim N(0, \Omega^w), \vartheta_k \sim N(0, \Omega^\vartheta), x_0 \sim N(x_0^-, \Lambda_0^-), I_0 = y_0. \tag{3.162}$$

Since, we assume that waypoints $\{b_{\alpha_1}, b_{\alpha_2}, ..., b_{\alpha_m}\}$ are along the best trajectory, the above problem is equivalent to the following optimization problem:

$$J^* = \min_{\Pi} E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k)\Big| I_0\right] \tag{3.163}$$

subject to:

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k\sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{3.164}$$

$$E\left[(x_{k_j} - x^s_{\alpha_j})' Q_{k_j} (x_{k_j} - x^s_{\alpha_j})\Big| I_0\right] \leq \epsilon_{k_j}, \quad j = 1, 2, ..., m \tag{3.165}$$

$$y_k = g(x_k, u_k) + \vartheta_k, \qquad k = 0, 1, ..., N \tag{3.166}$$

$$I_k = [y_0, y_1, ..., y_k, u_0, u_1, ..., u_{k-1}], \tag{3.167}$$

$$u_k = \mu_k(I_k), \tag{3.168}$$

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{3.169}$$

$$x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, y_k \in \mathbb{R}^{n_y}, w_k \in \mathbb{R}^{n_w}, \vartheta_k \in \mathbb{R}^{n_\vartheta}, \tag{3.170}$$

$$w_k \sim N(0, \Omega^w), \vartheta_k \sim N(0, \Omega^\vartheta), x_0 \sim N(x_0^-, \Lambda_0^-), I_0 = y_0. \tag{3.171}$$

Eq. 3.165, with symmetric matrices $Q_{i_j} \succ 0$, represents m waypoints along a globally optimal trajectory. Assume that $h(x_N) = (x_N - x_G)'Q_N(x_N - x_G)$, we denote $h_{k_j}(x_{k_j}) = (x_{k_j} - x^s_{\alpha_j})'Q_{k_j}(x_{k_j} - x^s_{\alpha_j})$, and Eq. 3.165 can be rewritten as

$$E\left[h_{k_j}(x_{k_j})\middle| I_0\right] \leq \epsilon_{k_j}, \quad j = 1, 2, ..., m. \tag{3.172}$$

The term ϵ_{k_j} represents the distance to the sampled mean of the j^{th} waypoint when the robot passes by. If $\epsilon_{k_j} = 0$, the robot is required to pass by exactly the sampled mean $x^s_{\alpha_j}$ along the planned trajectory α. These margin regions allow the robot to perform a smooth trajectory around the planned trajectory α. Dualizing this set of constraints yields the following dual problem:

$$\mathcal{J}(\lambda) = \min_{\Pi} E\left[h(x_N) + \sum_{k=0}^{N-1} \ell_k(x_k, u_k) + \sum_{j=1}^{m} \lambda_j\left(h_{k_j}(x_{k_j}) - \epsilon_{k_j}\right)\middle| I_0\right] \tag{3.173}$$

subject to:

$$x_{k+1} = x_k + f(x_k, u_k)\Delta + w_k\sqrt{\Delta}, \qquad k = 0, 1, ..., N-1 \tag{3.174}$$

$$y_k = g(x_k, u_k) + \vartheta_k, \qquad k = 0, 1, ..., N \tag{3.175}$$

$$I_k = [y_0, y_1, ..., y_k, u_0, u_1, ..., u_{k-1}], \tag{3.176}$$

$$u_k = \mu_k(I_k), \tag{3.177}$$

$$\pi = \{\mu_0, \mu_1, ..., \mu_{N-1}\}, \tag{3.178}$$

$$x_k \in \mathcal{X} \subseteq \mathbb{R}^{n_x}, u_k \in \mathcal{U} \subseteq \mathbb{R}^{n_u}, y_k \in \mathbb{R}^{n_y}, w_k \in \mathbb{R}^{n_w}, \vartheta_k \in \mathbb{R}^{n_\vartheta}, \tag{3.179}$$

$$w_k \sim N(0, \Omega^w), \vartheta_k \sim N(0, \Omega^\vartheta), x_0 \sim N(x_0^-, \Lambda_0^-), I_0 = y_0. \tag{3.180}$$

The vector $\lambda = [\lambda_1, \lambda_2, ..., \lambda_m]'$ represents m dual variables corresponding to m dualized constraints. We note that for a given λ, the term $\sum_{j=1}^{m} \lambda_j \epsilon_{k_j}$ is a constant. Thus, in the final run of iLQG algorithm, a returned control law achieves the objective value $\mathcal{J}(e)$, where $\lambda = e = [1, 1, ..., 1]'$. From the weak duality Theorem, the lower-bound of the optimal value of the original problem is given by:

$$\mathcal{J}(e) \leq \max_{\lambda \geq 0} \mathcal{J}(\lambda) \leq J^*. \tag{3.181}$$

On the other hand, the upper bound of J^* can be obtained by any admissible control law to reach a destination. In particular, a suggested value from the Dijkstra's search to traverse edges in a sub-optimal trajectory can be used as an upper bound. Thus, we have:

$$\mathcal{J}(e) \leq J^* \leq J^{Dijkstra}. \tag{3.182}$$

Eq. 3.182 provides us with the bounds of the globally optimal value of the original problem, in which the bound limits are found by the combination of the iLQG algorithm and the construction and search of a belief graph. Therefore, in principle, if error from every step in the entire POBRM algorithm as discussed in Section 3.6 are bounded, we will able to devise the POBRM algorithm to verify this bound experimentally.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Experiments And Results

In this chapter, we present several experiments and results to verify the performance of the POBRM algorithm. In particular, the experiments were tested on simulation of the modeled 2D plannar car and the 3D helicopter in Chapter 2. First, we show sub-optimal trajectories to follow edges generated by the iLQG algorithm. The performances of covariance transfer functions and cost transfer functions are then presented. We provide sub-optimal trajectories resulting from planning and controlling to navigate from a starting location to a destination. Finally, we show that the algorithm is scalable from 2D systems to complex 3D systems.

4.1 Performance of the iLQG algorithm

Table 4.1 summarizes the landmarks locations and their visibility radii, system noise covariance Ω^w, measurement noise covariance Ω^ϑ, and initial pose covariance Λ_0^-. We assume that there is no state and control constraint for this system. Figure 4-1 shows an example of an iLQG trajectory for the 2D car. In this figure, the iLQG algorithm produces a sub-optimal trajectory to traverse an edge connecting the green star and the red star. The features are red dots, and the dashed blue circles around the features are the visibility areas for the car to observe the corresponding features. There are three trajectories plotting in this figure: the nominal trajectory in blue, the true trajectory in red, and the estimated trajectory in green. Around the estimated

Table 4.1: An example of settings in an environment

Landmarks	Radius (m)
$(-15, -3)$	7
$(2, 13)$	7
$(14, 12)$	5
$(3, -22)$	8
$(20, -12)$	4

$$\Omega^w = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \quad \Omega^\vartheta = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.002 \end{bmatrix}$$

$$\Lambda_0^- = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.001 \end{bmatrix}$$

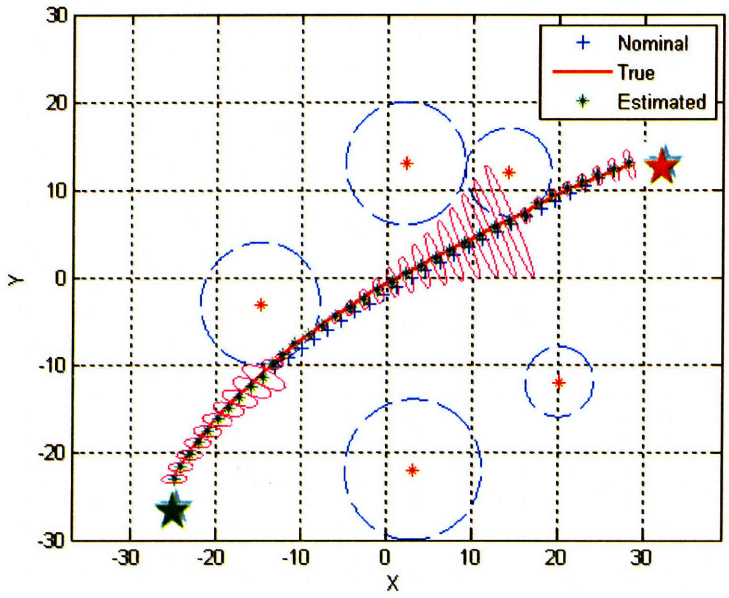


Figure 4-1: An example of iLQG trajectory.

trajectory, there are ellipses representing the covariances of these estimates. The covariances become larger over time if the car is not nearby some visibility area. When the covariances are large, the location of the car becomes highly uncertain, which will affect subsequent locations and the final pose in different realizations. However, as the iLQG algorithm is a CE controller, it does not take into account the minimization of the covariances along a trajectory. Therefore, this figure suggests that a high level planner should make use of available features to enhance the confidence of the car.

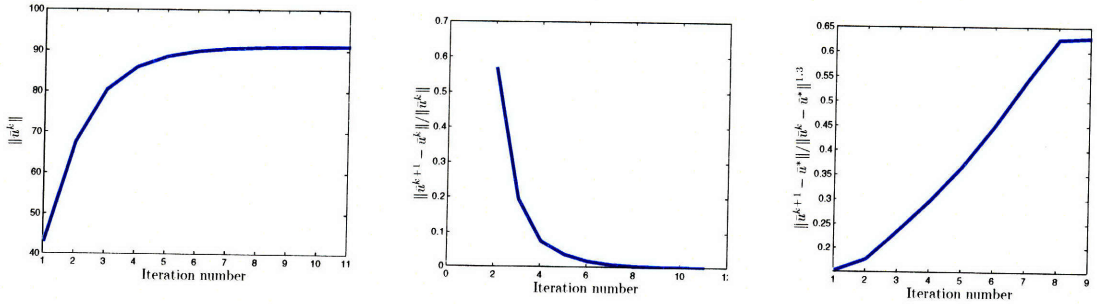
Figure 4-2 plots the profile of nominal controls $\bar{u}$ over 11 iterations for the con-

verged trajectory in Figure 4-1 with $\rho = 0.5$. Part 4-2(a) shows the norms of nominal control sequences, part 4-2(b) shows the ratio of improvement of nominal controls to their absolute norms over iterations in the stopping criteria, and part 4-2(c) presents the plot that verifies the rate of convergence. In part 4-2(c), we calculated ratios $\frac{\|\bar{u}^{k+1}-\bar{u}\|^*}{\|\bar{u}^k-\bar{u}^*\|^{1.3}}$ where $\bar{u}^*$ is the converged nominal control sequence. As all the ratios are in the range $(0, 1)$, it indicates that the rate of convergence is about 1.3 in this case. We take note that when updating a new nominal control sequence, the first step in Theorem 3.3.1 is a Newton-like method, which has a quadratic rate of convergence. Then the new control sequence is interpolated with the old control sequence via the parameter ρ to avoid arbitrary diverged simulated control sequences due to noise. Thus, the rate of convergence is less than 2 as we can expect.

4.2 Performance of transfer functions

4.2.1 Covariance transfer functions

In the second experiment, we compared estimated covariances with fully updated EKF covariances for final state estimates at incoming vertices. Random covariance matrices with standard deviations for the X, Y directions up to 1 meter were given at a departing vertex. Corresponding estimated covariances and fully updated EKF covariances were recorded. Figure 4-3 depicts the traces of estimated covariances versus the traces of fully updated covariances. As we can see, different initial de-



(a) Plot of $\bar{u}$.

(b) Plot of change of $\bar{u}$.

(c) Plot of rate of convergence.

Figure 4-2: Profile of nominal controls.

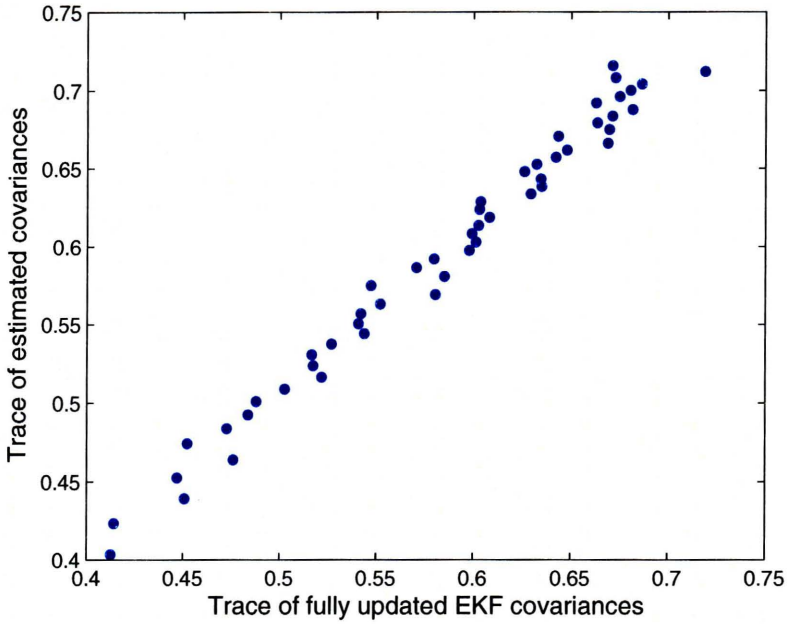


Figure 4-3: Estimated covariances v.s. fully updated covariances.

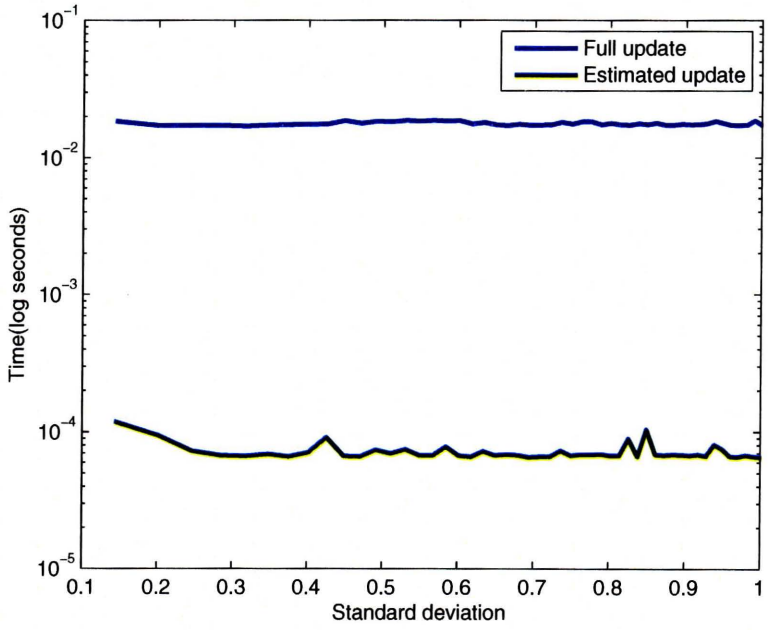


Figure 4-4: Running time to compute final covariances.

parting covariances yield different final covariances, and the trace values of the final covariances are closely matched by the two methods. Moreover, Figure 4-4 shows the running time, in the log scale, to compute the final covariances with respect to standard deviations in X, Y directions using the two methods. The fully updated

time of covariance matrices is computed averagely using 60 simulations. The graph shows that the fully updated time is almost 150 times more than the estimated time using transfer functions. This observation verifies that variance transfer functions are able to preserve the accuracy of predicted covariances with less intense computation.

4.2.2 Cost transfer functions

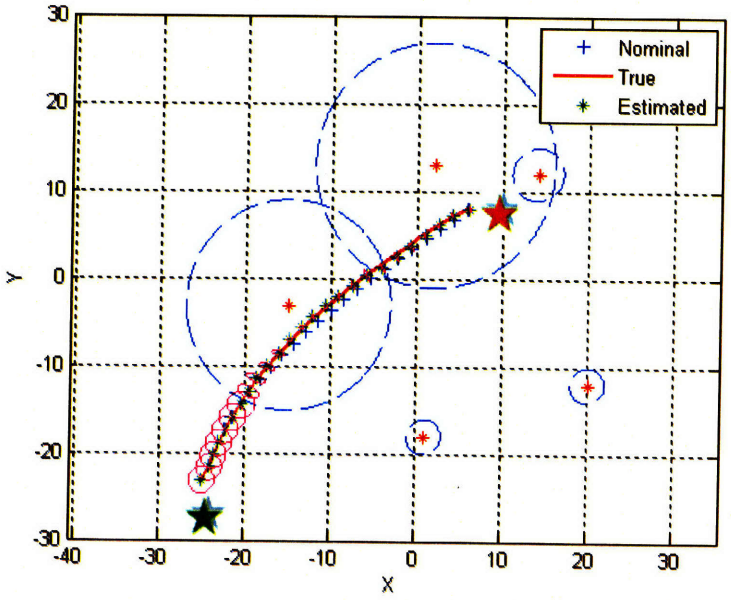


Figure 4-5: An example of an edge receiving many observations.

Next, we evaluated the performance of cost transfer functions for estimating the cost-to-go values for edges. Similar to covariance transfer functions, different initial covariances were given, we recorded both estimated cost values and average simulated cost values over 600 samples. Figure 4-5 shows an example of an edge along which the car will receive many observations. Figure 4-6 compares two types of cost values versus standard deviations for X, Y directions. Similarly, Figure 4-7 presents an edge without nearby landmarks and the corresponding comparison is shown in Figure 4-8. As suggested by these plots, we infer that the proposed approximation structure $\widetilde{r}_0 + tr(\widetilde{P}\Lambda)$ is able to preserve the shape and trend of objective values with respect to initial covariances. The accuracy of estimated values by cost transfer functions depends on noises and the structures of environments.

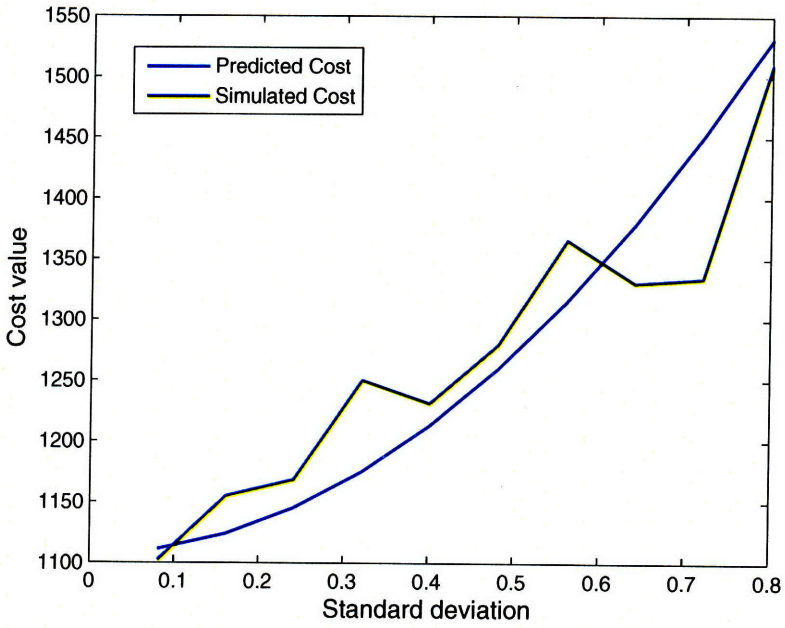


Figure 4-6: Estimated cost values v.s. simulated cost values for Figure 4-5.

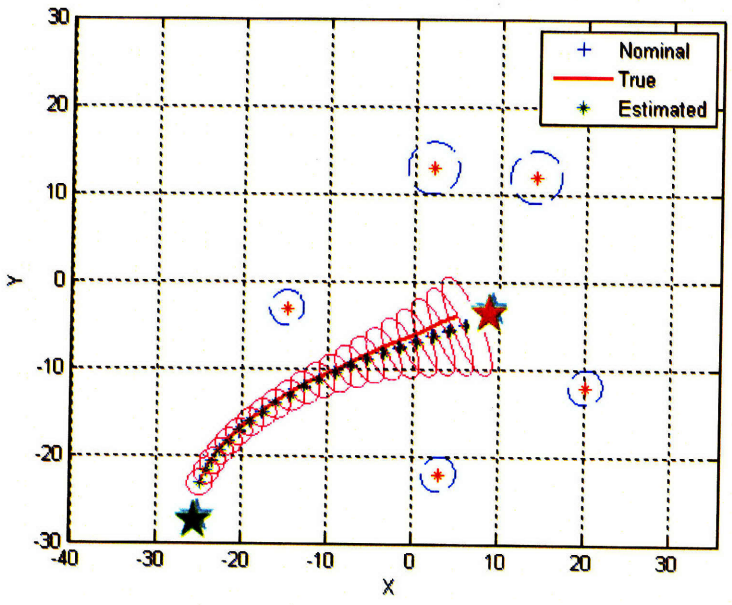


Figure 4-7: An example of an edge without any observation.

4.3 Planning and control

In the third experiment, we planned a mission to navigate from a starting location to a destination. There were five features, and a belief graph with ten sampled vertices was created in the offline phase. The online phase returned a sub-optimal trajectory and a locally-feedback control law to follow this trajectory. Figures 4-9 and 4-10

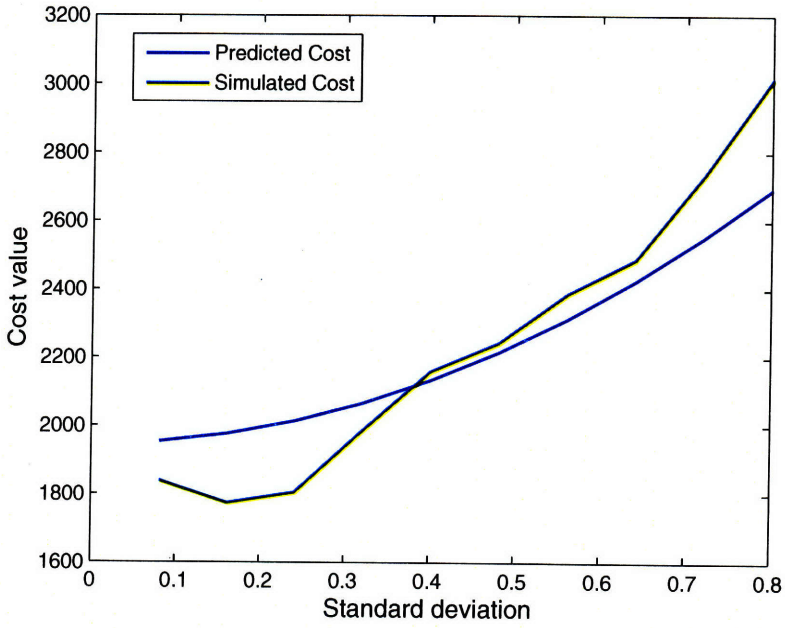


Figure 4-8: Estimated cost values v.s. simulated cost values for Figure 4-7.

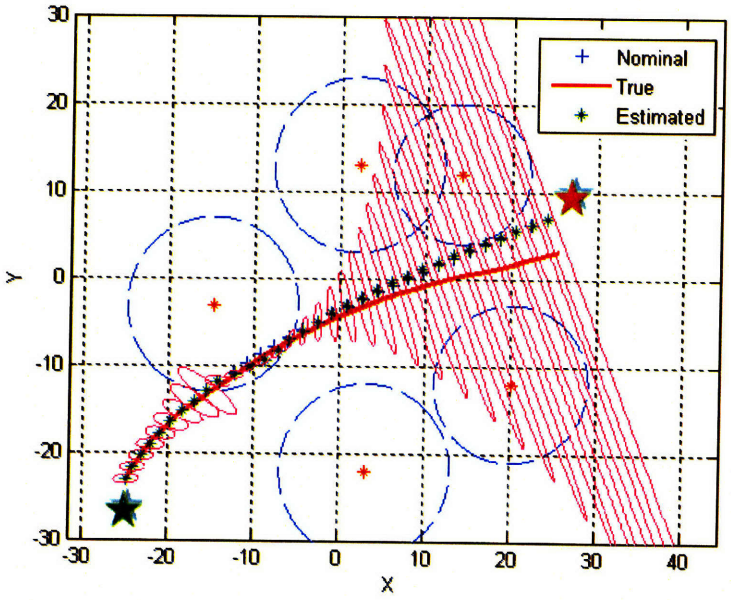


Figure 4-9: No waypoint, cost value 16735.

show the advantages of the POBRM algorithm in planning and controlling the car. On the first hand, Figure 4-9 plots a trajectory between two locations without going through any waypoints to reduce covariances. Thus, although the estimated mean of the final state is at the destination, the actual position of the car is far away from the destination. On the second hand, the POBRM algorithm provides the planned

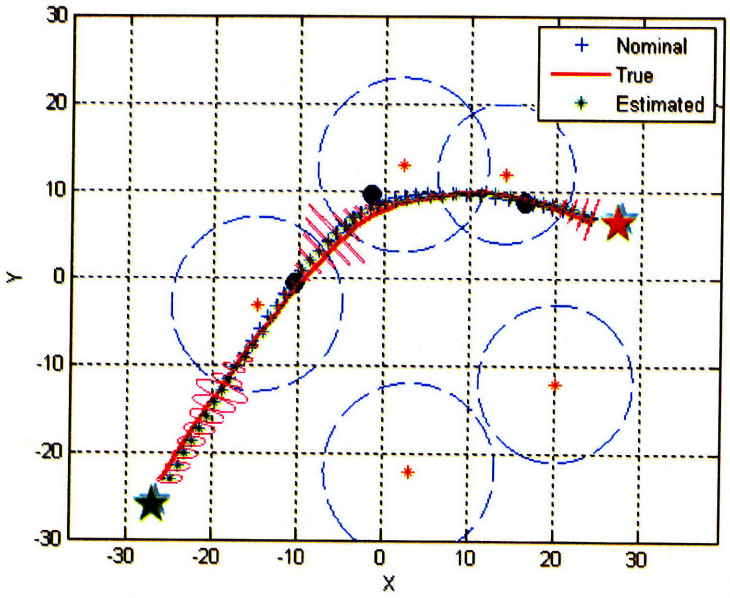


Figure 4-10: Three waypoints, cost value 340.

Table 4.2: Comparing of unplanned and planned trajectory costs.

Standard deviation (m)	Unplanned cost	Planned cost
$\sqrt{0.1}$	1.74×10^4	370
$\sqrt{0.2}$	2.03×10^4	394
$\sqrt{0.3}$	2.11×10^4	408
$\sqrt{0.4}$	2.22×10^4	434
$\sqrt{0.5}$	2.42×10^4	445

sub-optimal trajectory with three black waypoints in visibility areas in Figure 4-10. Therefore, the car is able to reach the destination with high accuracy despite the long curved route.

In Table 4.2, we compare the average simulated cost values of the two trajectories with varying standard deviations in X,Y direction. As we can see, the planned trajectory cost from the POBRM algorithm is significantly smaller than the unplanned trajectory cost. Hence, this result infers that more energy to traverse the longer trajectory benefits the car tremendously. In addition, when the standard deviations

Table 4.3: Running time requirement for the offline and online phases.

Offline time (s)	Online time (s)
600.43	1.32

increase, the simulated cost of the planned trajectory increases slightly, which indicates that the POBRM algorithm is robust against initial state uncertainty.

Table 4.3 summarizes the running time in seconds to build the belief graph in the offline phase, and the searching in the online phase. As we can see, compared to the online time, the offline time is almost 455 times slower, which is substantially dominant in the POBRM algorithm. Thus, most of the burden of computation is moved to the offline phase.

4.4 Scalability to high-dimensional systems

Finally, we verified that the algorithm is scalable to planning and controlling the helicopter in 3D environments. In this case, the problem has control constraints on the thrust components of control inputs. We investigated three important operations in controlling the helicopter: hovering, landing, and navigating.

4.4.1 Hovering

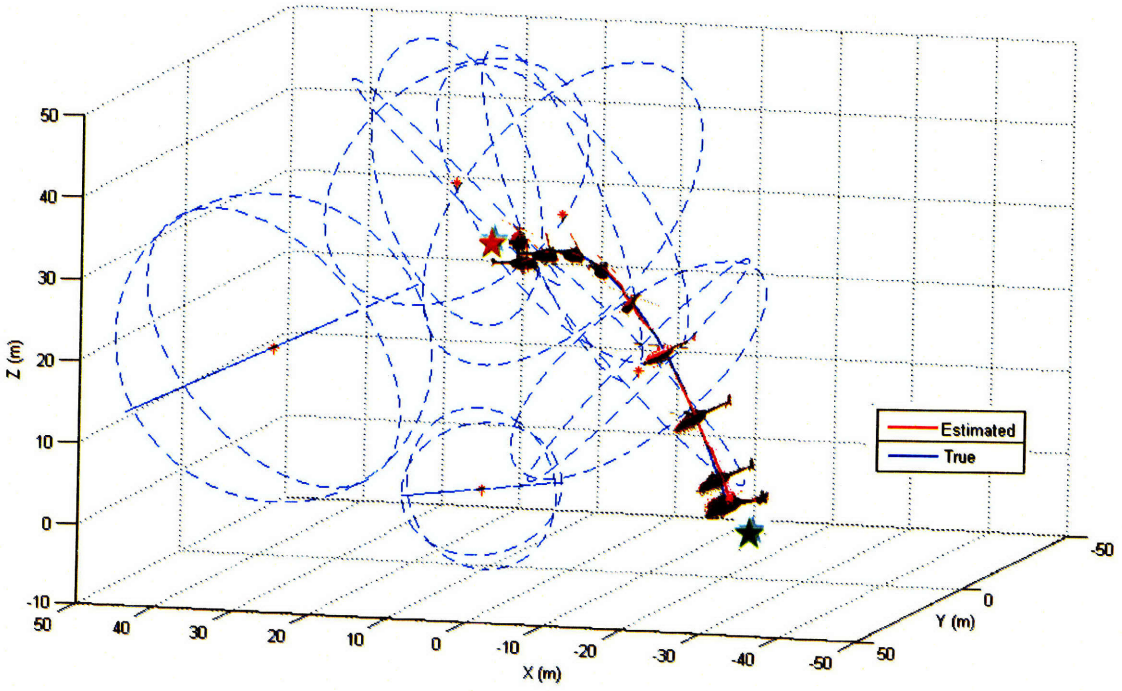


Figure 4-11: An example of the hovering task.

In controlling the helicopter, one of the most common tasks is to hover around a position after taking off from the ground. Figure 4-11 shows an example of this task. Similar to the 2D planar car, there are landmarks as red dots, and the dashed spheres around them are visibility areas. As shown in this figure, the helicopter is able to take off and then hovers around the red star position for a while.

4.4.2 Landing

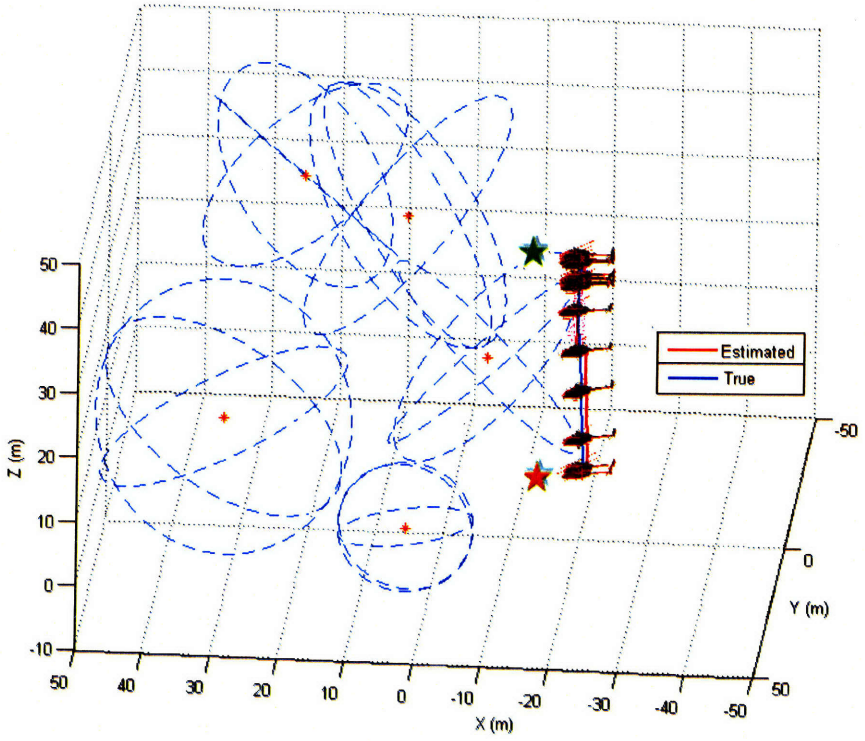


Figure 4-12: An example of the landing task.

The next common task is to land safely on the ground ($Z = 0$). Figure 4-12 depicts an example of landing trajectories. As we can see, the helicopter approaches the ground without overshooting negative Z values.

4.4.3 Navigating

Similar to 2D planar car navigating missions, we used the POBRM algorithm to plan and navigate the helicopter to a specified destination. We compared the effect of

trajectory planning and control using the iLQG algorithm alone and using POBRM. In Figure 4-13, we present an example of an unplanned trajectory, which is returned from the iLQG algorithm. As we can see, the covariances, which are represented by the dotted cloud around each estimated mean, are enormous. Thus, the helicopter is uncertain about reaching the desired destination. In contrast, in Figure 4-14, a planned sub-optimal trajectory with two waypoints in black, which is returned from the combination of the Dijkstra's search in the belief graph and the iLQG algorithm, is used. This trajectory arrives at the destination with high accuracy by taking the advantage of the necessary nearby features.

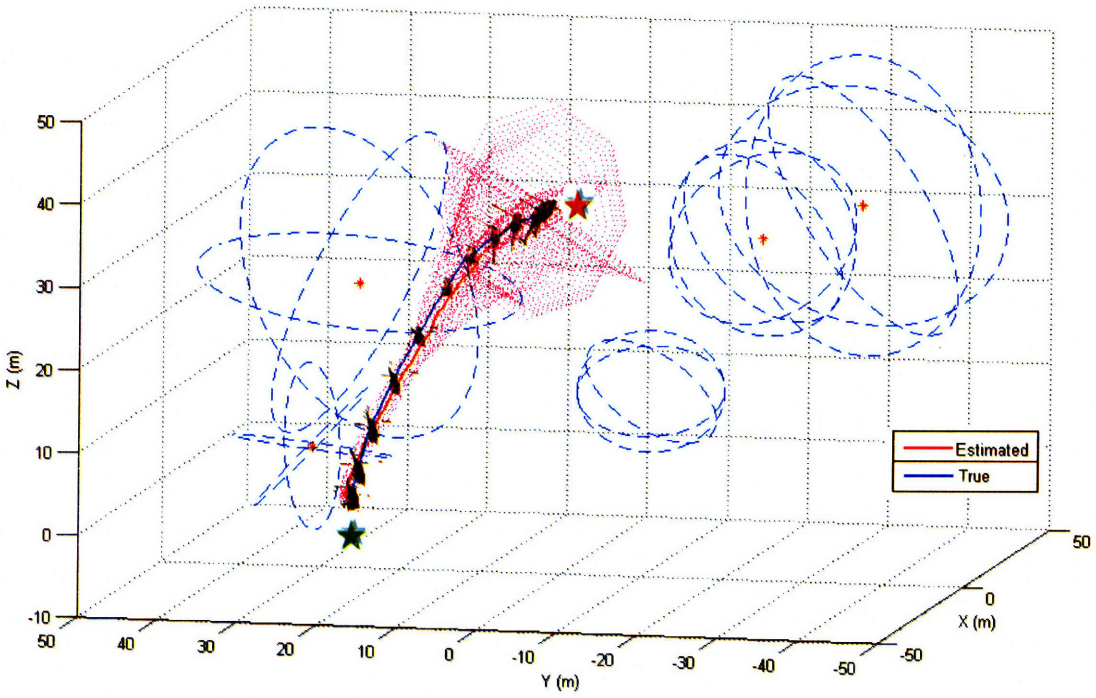


Figure 4-13: A trajectory using the iLQG algorithm only with large covariances.

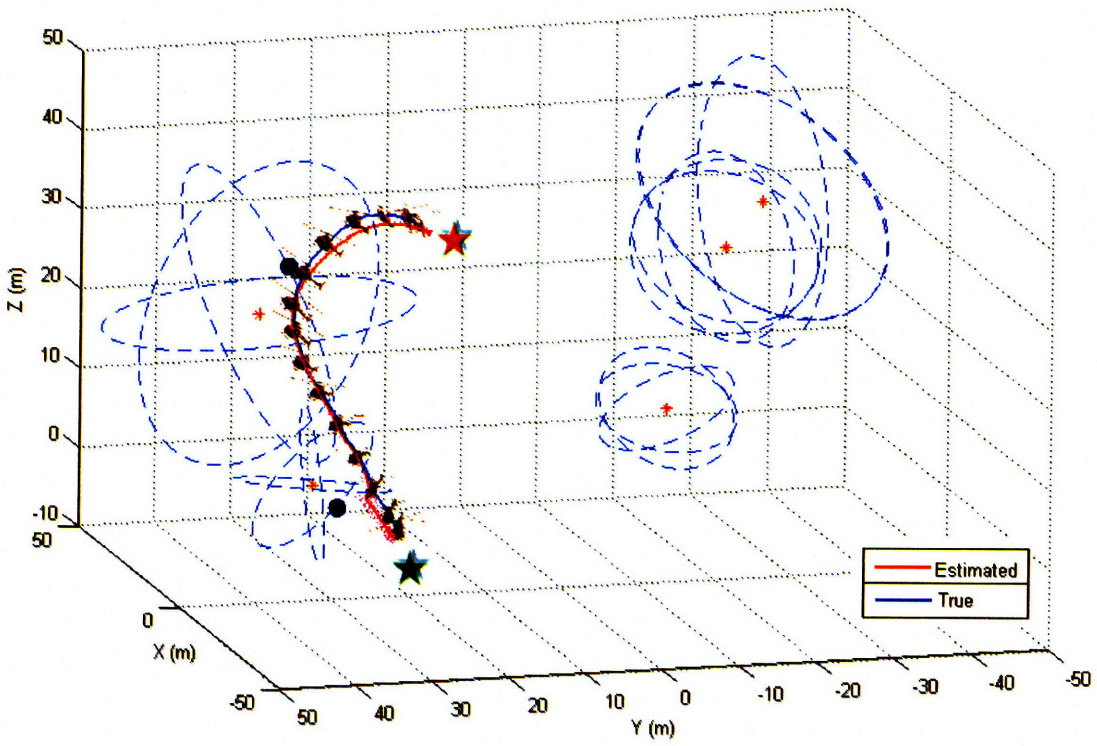


Figure 4-14: A trajectory with the iLQG algorithm and the Dijkstra's search.

Chapter 5

Conclusions

5.1 Summary

In this research, we have addressed the problem of planning and control of mobile robots in partially observable stochastic environments, when state estimates use feature-based map information. The POBRM algorithm is inspired by the PRM algorithm [10], which is highly successful in planning trajectories for fully observable robots. We have incorporated the idea of constructing belief graphs in the BRM algorithm [22] [8]. The key concept is to sample in the mean space and search in the covariance space of robots' belief states. To plan a trajectory efficiently, each edge of the belief graph is associated with a covariance transfer function and a cost transfer function. These functions parameterize the set of different trajectories based on covariances of initial beliefs. Thus, the POBRM algorithm is able to move significant computation to the offline phase, and perform fast planning in the online phase. Moreover, the POBRM algorithm also provides a sub-optimal control policy to control robots along a planned trajectory using the iLQG algorithm [15]. The principle of the iLQG algorithm is based on solving successive linear quadratic Gaussian problems by the dynamic programming. Overall, the POBRM algorithm can be regarded as a POMDP method to solve the partially observable stochastic shortest path problem.

From our proposed approaches and results, this work has contributed many as-

pects in the field of robust robotics. We have proposed a novel method to decouple computation in two separate phases that will facilitate fast real-time planning and controlling. The algorithm takes into account of balancing exploitation in terms of minimizing energy and exploration in terms of increasing confidence. Thus, the POBRM algorithm is robust to imperfect state information. Moreover, we have provided an insight into the relationship among different subproblems from the view of dualizing complex constraints. This leads to a potential theoretical error bound analysis in future work. Finally, we have demonstrated that the proposed algorithm is scalable from 2D planar car systems to 3D helicopter systems to perform coastal navigation trajectories.

5.2 Future directions

There are several directions to continue this research in the future. First, we have been working so far on simulation. Thus, the next step would be to verify the POBRM algorithm on real hardware. We are under the process of building the dynamics for the quad-helicopter and going to test autonomous indoor flight.

Second, the next challenge is to handle state constraints and control constraints in the iLQG algorithm more rigorously. The current heuristic approach suffers from slow convergence if the set of state and control constraints is too complex. Moreover, the assumption of converting from state constraints to control constraints is not always feasible due to complex system dynamics. There are some possible directions to tackle this issue. Based on the idea of embedding complex constraints into the objective function, the well known dualizing [2] and barrier methods [2] in nonlinear optimization paradigm are suitable in our circumstance. As an alternative approach, methods based on algebraic geometry [7] are likely to be useful. Thus, more necessary investigations need to be carried out to fully handle these complex constraints.

Third, a more systematic way to sample vertices in a belief graph is desirable to reduce the number of samples and at the same time ensure good information. This is highly important for navigating within large-scale indoor environments.

Fourth, most of the offline phase of the POBRM algorithm is spent in building cost transfer functions. More work to accelerate this slow learning process is essential.

Last but not least, a more rigorous error analysis of all steps in the POBRM algorithm is highly essential to ensure the stable performance of the algorithm. Especially, we have to scrutiny the time discretization step, the iLQG algorithm, the sampling belief graph step, the construction of transfer functions, the Dijkstra's search step, and the final iLQG pass. The proven error bounds of all these parts will enable us to understand the error bound of the original continuous-time problem.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] Yaakov Bar-Shalom. Stochastic dynamic programming: Caution and probing. *IEEE Transactions on Automatic Control*, 26(5):1184–1195, 1981.

[2] Dimitri P. Bertsekas. *Nonlinear Programming.* Athena Scientific, September 2003.

[3] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I–II. Athena Scientific, PO Box 391, Belmont, MA 02178, 2007.

[4] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming (Optimization and Neural Computation Series, 3).* Athena Scientific, May 1996.

[5] Blai Bonet. An epsilon-optimal grid-based algorithm for partially observable markov decision processes. In *ICML*, pages 51–58, 2002.

[6] Ronen I. Brafman. A heuristic variable grid solution method for POMDPs. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 76–81, Providence, Rhode Island, 1997. AAAI Press / MIT Press.

[7] I. A. Fotiou, P. Rostalski, P. A. Parrilo, and M. Morari. Parametric optimization and optimal control using algebraic geometry methods. *International Journal of Control*, 79(11):1340–1358, 2006.

[8] Ruijie He, Sam Prentice, and Nicholas Roy. Planning in information space for a quadrotor helicopter in a gps-denied environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2008)*, pages 1814–1820, Los Angeles, CA, 2008.

[9] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Waslander, and Claire J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Hilton Head, South Carolina, August 2007.

[10] Lydia Kavraki, Petr Svestka, Jean Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[11] H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*, 2008.

[12] Jack Langelaan and Steve Rock. Towards Autonomous UAV Flight in Forests. *2005 AIAA Guidance, Navigation, and Control Conference and Exhibit*, pages 1–13, 2005.

[13] Weiwei Li and Emanuel Todorov. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the American Control Conference*, pages 300–306, Portland, Oregon, June 2005.

[14] Weiwei Li and Emanuel Todorov. Iterative optimal control and estimation design for nonlinear stochastic systems. In *Proceedings of the 45th IEEE Control and Decision Conference*, San Diego, California, December 2006.

[15] Weiwei Li and Emanuel Todorov. Iterative linearization methods for approximately optimal control and estimation of non-linear stochastic systems. *International Journal of Control*, 80:1439–1453, 2007.

[16] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning*, pages 362–370, San Francisco, CA, USA, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[17] William S. Lovejoy. Computationally feasible bounds for partially observed markov decision processes. *Operations Research*, 39(1):162–175, 1991.

[18] Ruben Martinez-Cantin, Nando de Freitas, Arnaud Doucet, and Jose Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems (RSS)*, Atlanta, Georgia, June 2007.

[19] Kevin Murphy. A survey of pomdp solution techniques, 2000.

[20] Andrew Y. Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Proceedings of International Symposium on Experimental Robotics*, Singapore, June 2004.

[21] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. Anytime point-based approximations for large pomdps. *Journal of Artificial Intelligent Research (JAIR)*, 27:335–380, 2006.

[22] Sam Prentice and Nicholas Roy. The belief roadmap: Efficient planning in linear pomdps by factoring the covariance. In *Proceedings of the 13th International Symposium of Robotics Research (ISRR)*, Hiroshima, Japan, November 2007.

[23] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties, 2004.

[24] Nicholas Roy. *Finding Approximate POMDP Solutions Through Belief Compression.* PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2003.

[25] Nicholas Roy, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Coastal navigation – robot motion with uncertainty. In *Proceedings of the AAAI Fall Symposium: Planning with POMDPs*, Orlando, Florida, October 1998.

[26] Guy Shani, Ronen I. Brafman, and Solomon Eyal Shimony. Forward search value iteration for pomdps. In Manuela M. Veloso, editor, *IJCAI*, pages 2619–2624, 2007.

[27] Trey Smith and Reid Simmons. Heuristic search value iteration for pomdps. In *Proceedings of Uncertainty in Artificial Intelligence*, Banff, Alberta, 2004.

[28] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for pomdps. *The International Journal of Robotics Research*, 24:195–220, 2005.

[29] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning).* The MIT Press, March 1998.

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

[31] Sebastian Thrun, Yufeng Liu, Daphne Koller, Andrew Y. Ng, Zoubin Ghahramani, and Hugh Durrant-Whyte. Simultaneous Localization and Mapping with Sparse Extended Information Filters. *The International Journal of Robotics Research*, 23(7-8):693–716, 2004.

[32] Emanuel Todorov. *Bayesian Brain: Probabilistic Approaches to Neural Coding*, chapter 12, pages 269–294. The MIT Press, Cambridge, Massachusetts, 2006.

[33] Edison Tse, Yaakov Bar-Shalom, and Lewis Meier. Wide-sense adaptive dual control for nonlinear stochastic systems. *IEEE Transactions on Automatic Control*, 18(2):98–108, 1973.

[34] John N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[35] Tao Wang, Michael Bowling, and Dale Schuurmans. Dual representations for dynamic programming and reinforcement learning. In *Proceedings of the 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, Honolulu, Hawaii, April 2007.

[36] Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Stable dual dynamic programming. In *Proceedings of Advances in Neural Information Processing Systems*, Vancouver, B.C., December 2007.

[37] Greg Welch and Gary Bishop. An introduction to the kalman filter.

[38] Rong Zhou and Eric A. Hansen. An improved grid-based approximation algorithm for POMDPs. In *IJCAI*, pages 707–716, 2001.

[39] Peter Zipfel. *Modeling and Simulation of Aerospace Vehicle Dynamics, Second Edition.* AIAA, April 2007.