Master of Science Thesis in Applied Mathematics
Department of Electrical Engineering, Linköping University, 2020

Multi-Hypothesis Motion Planning under Uncertainty using Local Optimization

Anja Hellander

Master of Science Thesis in Applied Mathematics

Multi-Hypothesis Motion Planning under Uncertainty using Local Optimization

Anja Hellander

LiTH-ISY-EX--20/5314--SE

Supervisor: **Kristoffer Bergman**
ISY, Linköpings universitet
Franz Hofmann
Saab Dynamics

Examiner: **Daniel Axehill**
ISY, Linköpings universitet

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

Motion planning is defined as the problem of computing a feasible trajectory for an agent to follow. It is a well-studied problem with applications in fields such as robotics, control theory and artificial intelligence. In the last decade there has been an increased interest in algorithms for motion planning under uncertainty where the agent does not know the state of the environment due to, e.g. motion and sensing uncertainties. One approach is to generate an initial feasible trajectory using for example an algorithm such as RRT* and then improve that initial trajectory using local optimization.

This thesis proposes a new modification of the RRT* algorithm that can be used to generate initial paths from which initial trajectories for the local optimization step can be generated. Unlike standard RRT*, the modified RRT* generates multiple paths at the same time, all belonging to different families of solutions (homotopy classes). Algorithms for motion planning under uncertainty that rely on local optimization of trajectories can use trajectories generated from these paths as initial solutions. The modified RRT* is implemented and its performance with respect to computation time and number of paths found is evaluated on simple scenarios. The evaluations show that the modified RRT* successfully computes solutions in multiple homotopy classes.

Two methods for motion planning under uncertainty, Trajectory-optimized LQG (T-LQG), and a belief space variant of iterative LQG (iLQG) are implemented and combined with the modified RRT*. The performance with respect to cost function improvement, computation time and success rate when following the optimized trajectories for the two methods are evaluated in a simulation study.

The results from the simulation studies show that it is advantageous to generate multiple initial trajectories. Some initial trajectories, due to for example passing through narrow passages or through areas with high uncertainties, can only be slightly improved by trajectory optimization or results in trajectories that are hard to follow or with a high collision risk. If multiple initial trajectories are generated the probability is higher that at least one of them will result in an optimized trajectory that is easy to follow, with lower uncertainty and lower collision risk than the initial trajectory. The results also show that iLQG is much more computationally expensive than T-LQG, but that it is better at computing control policies to follow the optimized trajectories.

Acknowledgments

I would like to begin by thanking my academic supervisor Kristoffer Bergman for all valuable feedback and suggestions along the way. I would also like to thank my industrial supervisor Franz Hofmann at Saab Dynamics for always being willing to share his knowledge, as well as my examiner Daniel Axehill at Linköping University.

I would like to direct a special thanks to Torbjörn Crona at Saab Dynamics for giving me the opportunity to write this thesis, as well as to his colleagues for the welcoming atmosphere.

Finally, I would like to thank my family and friends for their constant support.

Linköping, June 2020
Anja Hellander

Contents

Notation

Abbreviations

Abbreviation	Explanation
EKF	Extended Kalman filter
KF	Kalman filter
LQG	Linear-quadratic-Gaussian
LQR	Linear-quadratic regulator
MDP	Markov decision process
NLP	Nonlinear programming
OBF	Obstacle barrier function
POMDP	Partially observable Markov decision process
RRT	Rapidly-exploring random tree

1 Introduction

The main topic for this thesis is motion planning under uncertainty using local optimization. The thesis was done at Saab Dynamics in Linköping. This chapter provides a short background and the objective of the thesis as well as research questions. Lastly, a brief outline of the thesis is given.

1.1 Background

Motion planning is an area with a wide range of applications within fields such as robotics, control theory, artificial intelligence and even computational biology [9]. A typical motion planning problem is to find a feasible path for an agent to follow that will take it from some starting point to a destination point while avoiding obstacles.

Traditional motion planners have assumed that the agent is capable of measuring its full state and that effects of control actions are completely deterministic. In practice, this is most often not the case. Measurements can be noisy, giving rise to sensing uncertainties. The effects of control actions may become stochastic due to unexpected or unmodelled external forces, leading to motion uncertainty. In the last decade there has been an increased interest in algorithms for motion planning that take these uncertainties under consideration when trying to compute a path. Rather than simply trying to find the shortest feasible path, a motion planner that considers uncertainties will try to lead the agent through areas where the most information can be gained through sensing and the least information is lost due to motion uncertainties in order to maximize the probability of the agent reaching its destination point.

One way of solving the motion planning under uncertainty problem is through

the use of the partially observable Markov decision processes (POMDP) framework. The agent is not able to directly observe its state due to, e.g., measurement uncertainties. Instead, it keeps track of its internal belief state, which is defined by the probability distribution over all possible states given previous observations and control inputs. An exact solution to a POMDP will give the optimal control policy for each possible belief state.

1.2 Objective

The objective of this work is to implement two different methods for motion planning that use local optimization, and combine them with a multi-hypothesis approach where multiple initial trajectories are generated.

This objective can be divided into the following two parts:

- To combine the RRT* algorithm with the concept of homotopy and implement an algorithm that generates multiple paths belonging to different families of solutions (homotopy classes).
- To implement and evaluate two different methods for motion planning under uncertainty that use local optimization, where the starting solutions for the optimization are generated from the initial paths computed by the algorithm above. The evaluation will be done in a simulation study, where the two methods are compared in several trajectory planning scenarios (in two dimensions).

1.3 Research questions

The aim of the thesis is to provide answers to the following questions:

1. What local optimization methods can be used for motion planning under uncertainty?
2. How can a feasible initial trajectory be computed, to be used as initial solution by the local optimization?
3. What performance can be expected when RRT* and homotopy are used to find the initial trajectories for the local optimization, in terms of computation time, improvement of initial trajectories and success rate when following the computed trajectories?

1.4 Outline

The outline of the thesis is:

- Chapter 2 gives an overview of some topics that are relevant to the thesis.
- Chapter 3 describes how initial trajectories can be generated and presents a modified version of RRT* that considers homotopy.
- Chapter 4 discusses the implementation details of the two motion planning methods.
- Chapter 5 presents the results of the simulation study.
- Chapter 6 contains a discussion of the results presented in previous chapters.
- Chapter 7 summarizes the main results and discusses what can be done in future work.

2
Theory

This chapter provides a theoretical background to some of the topics covered in this thesis.

- Section 2.1 presents the underlying framework of partially observable Markov decision processes.
- Section 2.2 gives an introduction to RRT and RRT*.
- Section 2.3 defines the concept of homotopy classes for trajectories.
- Section 2.4 describes Dubins paths, that can be used for computing initial trajectories for some vehicle models.
- Section 2.5 contains a description of the extended Kalman filter used for state estimation.
- Section 2.6 describes the two different methods for motion planning under uncertainty that are the focus of this thesis.

2.1 Partially observable Markov decision processes

Partially observable Markov decision processes (POMDP) is a mathematical framework used to model the interaction between an agent and its environment [20]. The environment is characterized by *state*. The state typically includes variables regarding the agent's position, orientation and velocity as well as variables regarding the location and features of surrounding objects. The state at time t is denoted $\mathbf{x_t}$.

The state is not known by the agent. Instead, the agent uses *measurements* or *observations* to obtain information about the state. The measurement at time t is denoted $\mathbf{z_t}$.

The state may change as a result of *control actions*. The control action taken by the agent at time t is denoted $\mathbf{u_t}$.

In the POMDP framework the state is only partially observable due to e.g. sensing uncertainties. The effects of control actions are considered stochastic rather than deterministic due to motion uncertainty. A POMDP must therefore include a model for the state transition probability $p(\mathbf{x_t}|\mathbf{x_{t-1}}, \mathbf{u_{t-1}})$ and the measurement probability $p(\mathbf{z_t}|\mathbf{x_t})$.

Since the agent cannot measure its state directly it will instead keep track of its *belief*. The agent's belief at time t is denoted $\mathbf{b}(\mathbf{x_t})$ or $\mathbf{b_t}$ and is the probability distribution of the state $\mathbf{x_t}$ given all past states control inputs $\mathbf{u_0}, \ldots, \mathbf{u_{t-1}}$ and all past measurements $\mathbf{z_1}, \ldots, \mathbf{z_t}$, i.e. $\mathbf{b}(\mathbf{x_t}) = p(\mathbf{x_t}|\mathbf{u_0}, \ldots, \mathbf{u_{t-1}}, \mathbf{z_1}, \ldots, \mathbf{z_t})$.

An exact solution to a POMDP should give an optimal control policy for each possible belief state. The problem of computing this has been shown to be PSPACE-complete and therefore unlikely to be solved efficiently [13].

One algorithm for finding a control policy is called *value iteration*. The aim of value iteration is to minimize the expected value of some user-defined cost functions that associate control actions and beliefs with costs. Let the time horizon (the index of the final step) be l, and let the cost functions be $c_l(\mathbf{b_l})$ and $c_t(\mathbf{b_t}, \mathbf{u_t})$. The objective is to find a control policy, i.e. a mapping $\pi : \mathbf{b} \rightarrow \mathbf{u}$, that determines the control action to take for any belief. Hence, the problem is to choose the control actions $\mathbf{u_t} = \pi(\mathbf{b_t})$ such that the following objective function is minimized:

$$\mathop{\mathbb{E}}_{\mathbf{z_1},\ldots,\mathbf{z_l}}[c_l(\mathbf{b_l}) + \sum_{t=0}^{t=l-1} c_t(\mathbf{b_t}, \mathbf{u_t})]. \tag{2.1}$$

Value iteration finds a solution to (2.1) through backward recursion:

$$v_l(\mathbf{b_l}) = c_l(\mathbf{b_l}) \tag{2.2}$$

$$v_t(\mathbf{b_t}) = \min_{\mathbf{u_t}} \; (c_t(\mathbf{b_t}, \mathbf{u_t}) + \mathop{\mathbb{E}}_{\mathbf{z_{t+1}}}[v_{t+1}(\mathbf{b_{t+1}})] \tag{2.3}$$

$$\pi_t = \operatorname*{argmin}_{\mathbf{u_t}} \; (c_t(\mathbf{b_t}, \mathbf{u_t}) + \mathop{\mathbb{E}}_{\mathbf{z_{t+1}}}[v_{t+1}(\mathbf{b_{t+1}})]. \tag{2.4}$$

Here $v_t(\mathbf{b_t})$ is called the *value function* at time t.

2.2 Rapidly-exploring random trees

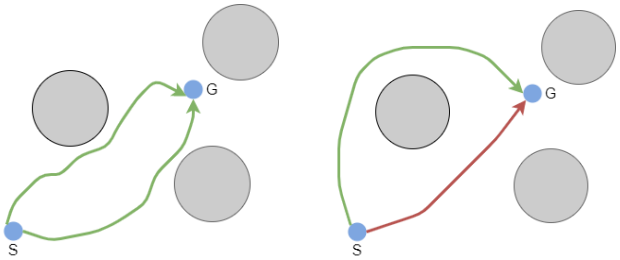
Rapidly-exploring random trees (RRT) is an algorithm used for path planning first described by Steven LaValle [8]. The algorithm is initialized with a tree that only consists of a single node that represents the initial state. In each iteration a random sample is drawn from the search space. The sampling can be biased toward the goal state, or it can be unbiased. The nearest neighbour of the sample, the node in the tree that is nearest to it, is determined. If the line between the two nodes does not intersect any obstacle and the distance between them is shorter than some pre-determined threshold d the node is added to tree with the neighbour as its parent. If the distance is greater than d, the new node is selected so as to lie on the line between the nearest neighbour and the random node, at distance d from the neighbour.

RRT* is a modified version of RRT that tries to find the path that minimizes some metric or cost, for example the Euclidean distance. Unlike RRT, RRT* has been proven to be asymptotically optimal, meaning that the cost of the returned path will converge to the optimum almost surely as planning time increases [7]. In RRT*, the parent node of a new node is chosen as the one that minimizes the cost of the new node, unlike in RRT where the closest node is always chosen. Another important difference between RRT* and RRT is that whenever RRT* has added a new node it performs so called *rewiring*. In the rewiring step, nodes that are sufficiently close to the newly added node are examined. Their current costs are compared to the costs they would have if the new node were their parent. If the new cost is lower, the node is rewired to have the new node as its parent (provided that the line between the nodes does not intersect any obstacle). In each step, the path from the start node to any other given node is the one with the lowest cost that can be found given the current nodes of the tree.

2.3 Homotopy

One way to classify different trajectories when obstacles are present is through the use of *homotopy classes*. A set of trajectories with the same start and end points is a homotopy class if the trajectories can be continuously deformed into one another without intersecting any obstacle [2]. An illustration of the concept of homotopy is seen in Figure 2.1.

Several different approaches to representing homotopy classes have been proposed. Two examples where homotopy class constraints have been combined with RRT or RRT* are homotopy-aware RRT* (HARRT*) [24] and homotopic RRT (HRRT) [4] that both construct a set of reference frames from lines that cut the obstacles. Paths can then be represented as a sequence of crossing between reference frames, represented as a string sequence. The approach used in this thesis, proposed by [2], is to associate each homotopy class with a complex-valued number computed through the use of the Cauchy integral. The approach is useful only for path planning in a plane where coordinates may be represented as points in the



(a) Two trajectories belonging to the same homotopy class.

(b) Two trajectories belonging to different homotopy classes, enclosing an obstacle.

Figure 2.1: *Illustration of homotopy classes. Two trajectories belonging to the same and to different homotopy classes.*

complex plane. The idea is that for each of the N obstacles a point ζ_i inside the obstacle is selected. A function $f_0(z)$ that is analytical everywhere is chosen, for example $f_0(z) = \sum_{i=1}^{N} z^i$. The function $F(z)$ is then chosen as $F(z) = \frac{f_0(z)}{(z-\zeta_1)\dots(z-\zeta_N)}$. $F(z)$ then has a single pole in $\zeta_0, \zeta_1, \dots, \zeta_N$ and is analytic elsewhere. For each path Γ from the start to the end point the Cauchy integral $\int_\Gamma F(z)dz$ can be computed. It follows from the Residue theorem that trajectories belonging to the same homotopy class correspond to the same integral value, whereas trajectories belonging to different homotopy classes correspond to different integral values. The integral value can therefore be used to represent an entire homotopy class. Note that the integral value is not unique as it depends on the choice of $f_0(z)$ and $\zeta_1, \dots, \zeta_N$. For some choices of $f_0(z)$ and $\zeta_1, \dots, \zeta_N$ it can happen that the integral value is the same for multiple homotopy classes. This can be avoided by defining

$$f_i(z) = (z - \zeta_i)F(z) = \frac{f_0(z)}{(z - \zeta_1)\dots(z - \zeta_{i-1})(z - \zeta_{i+1})\dots(z - \zeta_N)}, \; i = 1, \dots, N, \tag{2.5}$$

and selecting $f_0(z)$ and $\zeta_1, \dots, \zeta_N$ such that

$$\sum_{u \in S} f_u(\zeta_u) \neq 0 \tag{2.6}$$

is satisfied for any $S \subseteq \{1, \dots, N\}$ [2].

2.4 Dubins paths

Consider a car-like agent in a two-dimensional plane moving with constant speed that can only move in the forward direction. The agent has minimum turning radius ρ. Suppose that the agent should move from an initial position and heading to a final position and heading. It was shown by Dubins [3] that the shortest such

Table 2.1: *Shortest distance between circle centers and the corresponding Dubins path types.*

Shortest path	Dubins path type
CR_iCR_f	RSR
CR_iCL_f	RSL
CL_iCR_f	LSR
CL_iCL_f	LSL

path will consist of no more than three path segments, where each segment is either a straight line or a circular arc of radius ρ. If the distance between the initial and final positions is sufficiently large, the shortest path will consist of an arc segment followed by a straight line followed by an arc segment. Denoting a straight line segment with S, a circular arc to the right with R, and a circular arc to the left with L the four possible path types can be denoted RSR, RSL, LSR and LSL. The four different path types are illustrated in Figure 2.2.

To determine which path type to use, [10] propose computing the centers of the four possible circles, CR_i, CL_i, CR_f and CL_f and comparing the distances between pairs of circle centers. The Dubins path is then chosen according to Table 2.1.

For details on how to compute the circle centers and corresponding Dubins paths, see [10].

2.5 Extended Kalman filter

The Kalman filter (KF) [6] can be used to estimate the state and covariance of some variables given a series of observed measurements. When a KF is used, all error distributions are assumed to be Gaussian. The original KF assumes a linear state transition model as well as a linear measurement model. Nonlinear systems can be handled by the extended Kalman filter (EKF).

The EKF assumes the following state transition and observation model

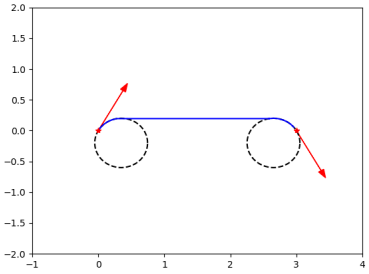
$$x_k = f(x_{k-1}, u_k) + w_k \tag{2.7}$$

$$z_k = h(x_k) + v_k, \tag{2.8}$$

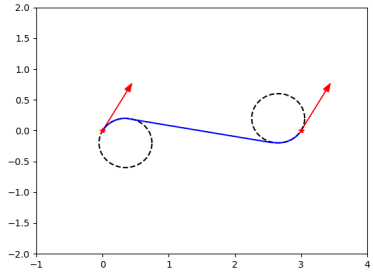
where $w_k \sim \mathcal{N}(0, Q_k)$ and $v_k \sim \mathcal{N}(0, R_k)$ are the process and observation noise respectively.

The state of the filter at time k is represented by $\hat{x}_{k|k}$, the a posteriori state estimate given observations up to and including time k, and $P_{k|k}$, the a posteriori estimate covariance matrix.

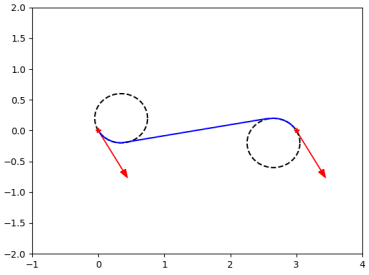
The EKF can be divided into two different phases: Predict and Update. In the predict phase a priori state and covariance estimates of the current time step are computed from the (a posteriori) estimates of the previous time step. The update



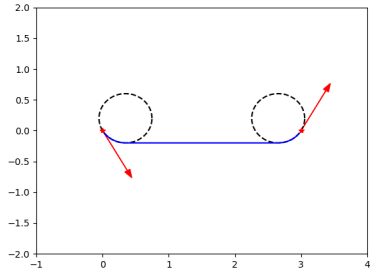
(a) RSR path.



(b) RSL path.



(c) LSR path.



(d) LSL path.

Figure 2.2: *Examples of Dubins curves. The initial and final positions are marked with red dots, with headings drawn as red arrows. The resulting Dubins curves are shown in blue.*

phase then uses the observation of the current time step together with the a priori estimates to compute the a posteriori state and covariance estimates. The steps of both phases are seen in (2.9)-(2.15). Here, $F_k = \frac{\partial f}{\partial x}|_{\hat{x}_{k-1|k-1}, u_k}$, $H_k = \frac{\partial h}{\partial x}|_{\hat{x}_{k|k-1}}$.

Predict

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \quad \text{Predicted (a priori) state estimate} \quad (2.9)$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \quad \text{Predicted (a priori) covariance estimate} \quad (2.10)$$

Update

$$\tilde{y}_k = z_k - h(\hat{x}_{k|k-1}) \quad \text{Innovation or measurement residual} \quad (2.11)$$

$$S_k = H_k P_{k|k-1} H_k^T + R_k \quad \text{Innovation covariance} \quad (2.12)$$

$$K_k = P_{k|k-1} H_k^T S^{-1} \quad \text{Near-optimal Kalman gain} \quad (2.13)$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k \quad \text{Updated state estimate} \quad (2.14)$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \quad \text{Updated covariance estimate} \quad (2.15)$$

2.6 Motion planning under uncertainty

Motion planning under uncertainty is an area that has seen an increased level of interest during the last decade. Previous work in the area include [15], [14], [19], [25]. The two methods that this thesis will focus on are *iLQG* [23] and *T-LQG* [17]. This section gives a brief introduction to their respective approaches to the motion planning problem. For more details, see [23] and [17].

2.6.1 Problem statement

The state-transition and observation models are

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{m}_t), \qquad \mathbf{m}_t \sim \mathcal{N}(0, I) \quad (2.16)$$

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{n}_t), \qquad \mathbf{n}_t \sim \mathcal{N}(0, I) \quad (2.17)$$

where m_t is the motion noise and n_t is the measurement noise.

As in Section 2.1, the belief $\mathbf{b}_t$ at time t is the probability distribution over all possible states. Both iLQG and T-LQG assume that this probability distribution is Gaussian distribution. Therefore, the belief $\mathbf{b_t} = (\hat{\mathbf{x}}_\mathbf{t}, \sqrt{\Sigma_t})$ at time t is defined by the mean $\hat{\mathbf{x}}_\mathbf{t}$ and principal square root $\sqrt{\Sigma_t}$ of the variance Σ_t of the Gaussian distribution $\mathcal{N}(\hat{\mathbf{x}}_\mathbf{t}, \Sigma_t)$. (In T-LQG, the representation used is $\mathbf{b_t} = (\hat{\mathbf{x}}_\mathbf{t}, \Sigma_t)$.) Since Σ_t is symmetric and positive semi-definite so is $\sqrt{\Sigma_t}$, and it is sufficient to represent either matrix using only the elements on or above the diagonal. If the dimension of the state $\mathbf{x_t}$ is n, the dimension of $\mathbf{b_t}$ will then be reduced to $\frac{n^2+3n}{2}$ compared to $n + n^2$ when the complete variance matrix is represented.

Given a start state and a goal state, the problem is to find a trajectory from the start to the goal that avoids obstacle collision, and an optimal control policy π :

$\mathbf{b} \rightarrow \mathbf{u}$ that can be used to follow the trajectory. As in the general POMDP case, the optimal control policy is the policy that minimizes the objective function of (2.1).

2.6.2 Iterative local optimization in belief space (iLQG)

One method for solving the motion planning under uncertainty problem is presented in [23], which will be denoted *iLQG*. The approach starts from an initial feasible trajectory $(\bar{\mathbf{b}}_0, \bar{\mathbf{u}}_0, \ldots, \bar{\mathbf{b}}_{l-1}, \bar{\mathbf{u}}_{l-1}, \bar{\mathbf{b}}_l)$. This initial trajectory is used as the first nominal trajectory. The belief dynamics are approximated using an EKF (see Section 2.5).

The method used to minimize the objective function in (2.1) is value iteration, described by (2.2)-(2.4). The value iteration is performed using a belief-space variant of the iterative LQG method developed by [21]. For each time step t the value function is approximated using a quadratic function that is locally valid in the vicinity of the nominal trajectory. The approximation is:

$$v_t(\mathbf{b}) \approx \frac{1}{2}(\mathbf{b} - \bar{\mathbf{b}}_t)S_t(\mathbf{b} - \bar{\mathbf{b}}_t) + (\mathbf{b} - \bar{\mathbf{b}}_t)\mathbf{s}_t + s_t. \tag{2.18}$$

For the final time step $t = l$, the approximation uses the following values, as described in [23]:

$$S_l = \frac{\partial^2 c_l}{\partial \mathbf{b}^2}(\bar{\mathbf{b}}_l), \qquad \mathbf{s}_l = \frac{\partial c_l}{\partial \mathbf{b}}(\bar{\mathbf{b}}_l), \qquad s_l = c_l(\bar{\mathbf{b}}_l). \tag{2.19}$$

The value iteration is performed backward in time, starting at $t = l$. For each time step t, the values of S_t, $\mathbf{s}_t$, s_t are computed recursively from S_{t+1}, $\mathbf{s}_{t+1}$, s_{t+1}. Detailed expressions are found in [23]. From the values of S_t, $\mathbf{s}_t$, s_t a control policy defined by $L_t, \mathbf{l}_t$ is computed. Detailed expressions are found in [23]. The control policy is

$$\mathbf{u}_t = \bar{\mathbf{u}}_t + L_t(\mathbf{b}_t - \bar{\mathbf{b}}_t) + \mathbf{l}_t. \tag{2.20}$$

In order to converge to a locally optimal solution, the nominal trajectory is updated using the computed control policy. The resulting trajectory is taken as the new nominal trajectory and the process is repeated until convergence.

Let the nominal trajectory of iteration i be denoted with $(\mathbf{b}_0^{(i)}, \mathbf{u}_0^{(i)}, \ldots, \mathbf{b}_l^{(i)})$ and its corresponding control policy with $L_t^{(i)}, \mathbf{l}_t^{(i)}$. The nominal trajectory of the next iteration, $(\mathbf{b}_0^{(i+1)}, \mathbf{u}_0^{(i+1)}, \ldots, \mathbf{b}_l^{(i+1)})$, is computed by starting at $\mathbf{b}_0^{(i+1)} = \mathbf{b}_0^{(i)}$ and applying the control policy forward in time. The control inputs are computed according to

$$\mathbf{u}_t^{(i+1)} = \mathbf{u}_t^{(i)} + L_t^{(i)}(\mathbf{b}_t^{(i+1)} - \mathbf{b}_t^{(i)}) + \mathbf{l}_t^{(i)}. \tag{2.21}$$

The beliefs of the new nominal trajectory are found using the EKF. Given the belief $\mathbf{b}_t = (\hat{\mathbf{x}}_t, \sqrt{\Sigma_t})$ and the control input $\mathbf{u}_t$ at time t, the belief $\mathbf{b}_{t+1} = (\hat{\mathbf{x}}_{t+1}, \sqrt{\Sigma_{t+1}}$

at time $t + 1$ is computed using the following equations [23]:

$$A_t = \frac{\partial f}{\partial \mathbf{x}}(\hat{\mathbf{x}}_\mathbf{t}, \mathbf{u}_\mathbf{t}, \mathbf{0}) \tag{2.22}$$

$$M_t = \frac{\partial f}{\partial \mathbf{m}}(\hat{\mathbf{x}}_\mathbf{t}, \mathbf{u}_\mathbf{t}, \mathbf{0}) \tag{2.23}$$

$$H_t = \frac{\partial h}{\partial \mathbf{x}}(f(\hat{\mathbf{x}}_\mathbf{t}, \mathbf{u}_\mathbf{t}, \mathbf{0}), \mathbf{0}) \tag{2.24}$$

$$N_t = \frac{\partial h}{\partial \mathbf{n}}(f(\hat{\mathbf{x}}_\mathbf{t}, \mathbf{u}_\mathbf{t}, \mathbf{0}), \mathbf{0}) \tag{2.25}$$

$$\Gamma_t = A_t\sqrt{\Sigma_t}(A_t\sqrt{\Sigma_t})^T + M_t M_t^T \tag{2.26}$$

$$K_t = \Gamma_t H_t (H_t \Gamma_t H_t + N_t N_t^T)^{-1} \tag{2.27}$$

$$\sqrt{\Sigma_{t+1}} = \sqrt{\Gamma_t - K_t H_t \Gamma_t}. \tag{2.28}$$

$$\hat{\mathbf{x}}_{t+1} = f(\hat{\mathbf{x}}_t, \mathbf{u}_t, \mathbf{0}) \tag{2.29}$$

Ensuring convergence using line search

In order to ensure convergence, line search is used [23]. This requires only a slight modification of the algorithm. A parameter ε is added to (2.21) and the control policy becomes

$$\mathbf{u}_t^{(i+1)} = \mathbf{u}_t^{(i)} + L_t(\mathbf{b}_t^{(i+1)} - \mathbf{b}_t^{(i)}) + \varepsilon \mathbf{l}_t. \tag{2.30}$$

Initially $\varepsilon = 1$ and each time the resulting trajectory does not decrease the expected cost ε is divided in half. If the resulting trajectory does result in a decreased expected cost ε is restored. One possible criterion for stopping the algorithm is when ε falls below a certain value. Another possible stopping criterion is to stop when the improvement in objective function value between two iterations is smaller than a certain value.

Collision avoidance

In order to avoid collision with obstacles the cost functions $c_t(\mathbf{b}_t, \mathbf{u}_t)$ should include some term that penalizes a high collision probability. The probability of colliding with an obstacle given a belief $\mathbf{b}_t = (\hat{\mathbf{x}}_t, \sqrt{\Sigma_t})$ is the integral of the probability density function of $\mathcal{N}(\hat{\mathbf{x}}_t, \sqrt{\Sigma_t})$ over the region of the state space that contains obstacles. This probability can be approximated as the number of standard deviations $\sigma(\mathbf{b}_t)$ that is possible to deviate from the mean before colliding with an obstacle [22]. A lower bound on the probability of not colliding is then $\gamma(\frac{n}{2}, \frac{\sigma(\mathbf{b}_t)^2}{2})$ where n is the dimension of the state space and γ is the regularized gamma function [23]. The collision avoidance function $f(\sigma(\mathbf{b}_t))$ is defined as

$$f(\sigma(\mathbf{b}_t)) = -\ln \gamma(\frac{n}{2}, \frac{\sigma(\mathbf{b}_t)^2}{2}). \tag{2.31}$$

The collision avoidance function, possibly multiplied with a (positive) scale factor can be added to the cost functions $c_t(\mathbf{b_t}, \mathbf{u_t})$ [23].

Cost function

The cost function used in the iLQG method is the cost function defined in (2.1). The immediate cost functions $c_l(\mathbf{b}_l)$ and $c_t(\mathbf{b}_t, \mathbf{u}_t)$ are chosen according to [23] as

$$c_l(\mathbf{b}_l) = (\hat{\mathbf{x}}_l - \mathbf{x}_{goal})^T Q_l (\hat{\mathbf{x}}_l - \mathbf{x}_{goal}) + \mathrm{tr}(\sqrt{\Sigma_l} Q_l \sqrt{\Sigma_l}) \tag{2.32}$$

$$c_t(\mathbf{b}_t, \mathbf{u}_t) = \mathbf{u}_t^T R_t \mathbf{u}_t + \mathrm{tr}(\sqrt{\Sigma_t} Q_t \sqrt{\Sigma_t}) + f(\sigma(\mathbf{b}_t)) \tag{2.33}$$

for matrices Q_l, Q_t, R_t. Here, $\mathbf{x}_{goal}$ is It is assumed that the Hessians of the cost functions c_l, c_t are positive (semi-)definite [23].

The first term in $c_l(\mathbf{b}_l)$ penalizes a deviation from the goal and the second term penalizes high uncertainty of state. The first term of $c_t(\mathbf{b}_t, \mathbf{u}_t)$ penalizes control effort, the second term penalizes uncertainty and the final term penalizes collision risk.

An approximation of the cost function can be computed during the value iteration. During the value iteration a control policy $L_t, \mathbf{l}_t$ is computed. The expected cost of the nominal trajectory that can be generated from this control policy is s_0 (see (2.18)) [23].

2.6.3 Trajectory-optimized LQG (T-LQG)

Trajectory-optimized LQG (T-LQG) as described by [17] propose designing the optimal trajectory and a control policy separately. This is in contrast to iLQG, where the computation of the optimal trajectory and control policy are closely linked. The problem of designing the optimal trajectory is in T-LQG modelled as a nonlinear programming (NLP) problem and can be solved using an NLP solver that takes a feasible initial trajectory as its initial guess.

Given an initial belief $\mathbf{b}_0 = (\hat{\mathbf{x}}_0, \Sigma_0)$ and a goal state $\mathbf{x}_{\text{goal}}$, the NLP problem to solve is

$$\min_{\mathbf{u}^p_{0:l-1}} \sum_{t=1}^{l} [\text{tr}(W_t \Sigma_t W_t^T) + (\mathbf{u}^p_{t-1})^T W_t^u (\mathbf{u}^p_{t-1})^T + c_{\text{obf}}(\mathbf{x}^p_t)] \tag{2.34}$$

$$\begin{aligned}
s.t.\ \mathbf{P}^p_{t|t-1} &= \mathbf{F}^p_t \mathbf{P}^p_{t-1|t-1} (\mathbf{F}^p_t)^T + \mathbf{Q}^p_t && \text{(a)}\\
\mathbf{S}^p_t &= \mathbf{H}^p_t \mathbf{P}^p_{t|t-1} (\mathbf{H}^p_t)^T + \mathbf{R}^p_t && \text{(b)}\\
\mathbf{P}^p_{t|t} &= (\mathbf{I} - \mathbf{P}^p_{t|t-1} (\mathbf{H}^p_t)^T (\mathbf{S}^p_t)^{-1} \mathbf{H}^p_t) \mathbf{P}^p_{t|t-1} && \text{(c)}\\
\mathbf{P}^p_{0|0} &= \Sigma_0 && \text{(d)}\\
\mathbf{x}^p_0 &= \hat{\mathbf{x}}_0 && \text{(e)}\\
\mathbf{x}^p_{t+1} &= f(\mathbf{x}^p_t,\ \mathbf{u}^p_t,\ \mathbf{0}),\ 0 \le t \le l-1 && \text{(f)}\\
\|\mathbf{x}^p_l - \mathbf{x}_{\text{goal}}\|_2 &< r_g && \text{(g)}\\
\|\mathbf{u}^p_t\|_2 &\le r_u,\ 0 \le t \le l-1, && \text{(h)}
\end{aligned}$$

where $W_t^T W_t = W_t^x$ and W_t^u are positive-definite (symmetric) weight matrices, and c_{obf} is a cost function that penalizes states with high collision probability. The function c_{obf} is described in more detail later in this section. Constraints (a)-(c) are regarded as one constraint at each time step [17] and describe the propagation of the covariance matrix Σ_t according to an EKF (see Section 2.5). Constraints (d) and (e) are initial conditions. Constraint (f) defines the state propagation (recall the state-transition model in (2.16)). Constraint (g) constricts the final state $\mathbf{x}^p_l$ to a ball of radius r_g around the goal state. Constraint (h) constricts the control inputs.

The optimal beliefs $\mathbf{b}^o_t$, $t = 1, \ldots, l$, are given by $\mathbf{b}^0_t = (\mathbf{x}^p_t, P^p_{t|t})$ from the solution to (2.34). The optimal control signals are given by $\mathbf{u}^o_t = \mathbf{u}^p_t$ from the solution to (2.34). The resulting optimal trajectory is $(\mathbf{b}^o_0, \mathbf{u}^o_0, \ldots, \mathbf{b}^o_l)$.

A linear-quadratic regulator (LQR) controller that follows the optimal trajectory is then designed after the optimal trajectory has been computed. The result is a series of feedback gains L^o_t and the control policy is

$$\mathbf{u}_t = \mathbf{u}^o_t - L^o_t(\hat{\mathbf{x}}_t - \mathbf{x}^o_t). \tag{2.35}$$

The feedback gains L^o_t are computed as

$$L^o_t = (W^u_t + B^T_t P_t B_t)^{-1} B^T_t P_t A_t, \tag{2.36}$$

where $A_t = \frac{\partial f}{\partial \mathbf{x}}(\hat{\mathbf{x}}^o_t, \mathbf{u}^o_t, \mathbf{0})$, $B_t = \frac{\partial f}{\partial \mathbf{u}}(\hat{\mathbf{x}}^o_t, \mathbf{u}^o_t, \mathbf{0})$, and P_t is computed recursively according to

$$P_{t-1} = A^T_t P_t A_t - A^T_t P_t B_t (W^u_t + B^T_t P_t B_t)^{-1} B^T_t P_t A_t + W^x_t, \tag{2.37}$$

with $P_l = W^x_t$ [16].

Note that while the control policy of iLQG is a function of the belief $\mathbf{b}_t$, the T-

LQG control policy is a function only of the state estimate $\hat{\mathbf{x}}_t$. The T-LQG also uses a different estimator to update the state and covariance estimates when following the trajectory. Instead of an EKF, the process model is linearized around the optimized trajectory and a KF is used. For details, see [16].

Collision avoidance

One way to handle obstacles is by adding an obstacle barrier function (OBF) $\Phi(\mathbf{x})$. Ideally, an OBF should be infinity for any $\mathbf{x}$ inside an obstacle and be zero elsewhere. In practice, an OBF should tend to infinity as $\mathbf{x}$ tends to an obstacle and take on low values outside of obstacles. To penalize collision, some cost function $c_{\text{obf}}(\mathbf{x}_t)$ that depends on the OBF is added to the optimization objective function. According to [17], one such cost function can be chosen according to:

$$c_{\text{obf}}(\mathbf{x}_t) = \int_{\mathbf{x}_{t-1}}^{\mathbf{x}_t} \Phi(\mathbf{x}')d\mathbf{x}'. \tag{2.38}$$

For more details on how $\Phi(\mathbf{x})$ can be chosen, see [17]. The special case when the obstacles are circular is treated in Section 4.2.1.

Cost function

The cost function, i.e. the objective function that is used in the optimization step is

$$\sum_{t=1}^{l} [\text{tr}(W_t \Sigma_t W_t^T) + \mathbf{u}_{t-1}^T W_t^u \mathbf{u}_{t-1} + c_{\text{obf}}(\mathbf{x}_t)] \tag{2.39}$$

where $W_t^T W_t = W_t^x$ and W_t^u are positive-definite (symmetric) weight matrices. The first term penalizes high uncertainty of state, the second penalizes control effort and the last term penalizes collision risk. This cost function is different from the iLQG cost function described in Section 2.6.2 in several ways. There is no expectation value, unlike in the iLQG cost function. There is no term that penalizes a deviation from the goal state, since that is represented by constraint (g) in the optimization problem. Finally, the term that penalizes high uncertainty of state is weighted the same for all time steps whereas the iLQG cost functions has a different weight for $t = l$.

3

Trajectory generation

This chapter describes the generation of one or several initial trajectories as a series of beliefs and control inputs $(\bar{\mathbf{b}}_0, \bar{\mathbf{u}}_0, \ldots, \bar{\mathbf{b}}_{l-1}, \bar{\mathbf{u}}_{l-1}, \bar{\mathbf{b}}_l)$.

- Section 3.1 presents a modified version of RRT* that considers homotopy, and describes how the algorithm is used to generate multiple paths represented as sets of waypoints. The algorithm is evaluated on two scenarios and the results are presented.
- Section 3.2 discusses how to generate an initial trajectory from a set of waypoints. The special case of trajectories based on Dubins paths is discussed in more detail.

3.1 Modified RRT* with homotopy

In order to find paths from the initial state to the goal state a modified version of RRT* is used. In this version, the search space is increased and includes the homotopy value (complex integral along the path) of each state. Since the homotopy value of a state depends on the homotopy value of previous states it cannot be be included in the sampling, since the parent node is not known at the time of sampling. Instead, a different approach is used. Unlike in the standard RRT* multiple nodes with the same coordinates are allowed, if the nodes have different parent nodes and belong to different homotopy classes. This allows the modified RRT* to find not only one but several paths from the start to the goal node.

The algorithm, described in Algorithm 1, starts with a tree consisting only of a node representing the initial state. In each iteration, a random node x_{rand} is sampled. As in standard RRT* the sampling is biased towards the goal node x_{goal}

with probability $0 < p < 1$. Then, the node x_{nearest} in the tree that is nearest to the x_{rand} is computed. If the sampled node x_{rand} is not equal to the x_{goal}, and if the path between x_{rand} and x_{nearest} is collision free, the steering function is called. The result of the steering function is a new node x_{new} with x_{nearest} as parent that added to the tree. As in standard RRT*, the steering function computes the distance between x_{rand} and x_{nearest}. If this distance is lower than some threshold *maxDistance*, x_{new} is selected as x_{rand}. If the distance is greater, x_{new} is selected so as to lie on the line between x_{rand} and x_{nearest}, at distance *maxDistance* from x_{nearest}. If the sampled node x_{rand} is equal to x_{goal}, instead of calling the steering function the node to be added to the search tree, x_{new}, is selected as x_{goal}. Next, the set $X_{\text{near}} = \{x | \, \|x - x_{\text{new}}\|_2 < \delta\}$ is computed. This set contains all nodes within distance δ of x_{new}. The parent of x_{new} is updated with one of the functions updateParent or updateGoalParent depending on whether x_{new} is equal to x_{goal} or not.

In standard RRT* the parent node of x_{new}, regardless of whether x_{new} is the goal or not, would be selected as

$$\min_{x \in X_{\text{near}}} \; cost(x) + cost(x, x_{\text{new}}). \tag{3.1}$$

In the modified version, updateParent and updateGoalParent work slightly different. The main difference is that the goal node can have multiple parents, where each one corresponds to a unique homotopy class. The updateGoalParent function loops through each node x in X_{near} and computes the potential homotopy class and cost of x_{goal} if x is the parent node of x_{goal}. If the associated cost is lower than the current best cost for that homotopy class, the parent node for x_{goal} for the homotopy class is updated. If the homotopy value has not been seen before, x is added to the list of parent nodes for x_{goal}.

The updateParent function starts by constructing a list *homotopyValues* that only contains the current homotopy value of x_{new}. Then, the function loops through each node x in X_{near}. For each x, the potential homotopy class value and cost that x_{new} would have if it had x as parent node is computed. If this homotopy value is not found in *homotopyValues*, a copy of x_{new}, but with x as parent is created and the homotopy value is added to the *homotopyValues*. If the homotopy value already exists in *homotopyValues* the cost is compared to the cost of the corresponding node. If it is lower than the previous one, that node will update its parent node. The function returns the set X_{new} consisting of all nodes (including x_{new}) that have been added.

After the updateParent function, rewiring is performed for each new node in X_{new}. The rewiring is similar to the one used in the standard RRT* algorithm, with the difference that a node may only update its parent as long as the homotopy value is not changed.

Algorithm 1 The modified RRT* algorithm

```
for i in range(maxIter) do
    x_rand ← getRandomNode()
    x_nearest, nearestDistance ← getNearestNeighbour(x_rand)
    if x_rand == x_goal then
        x_new ← x_goal
    else
        x_new ← steer(x_nearest, x_rand, maxDistance)
    end if
    if nearestDistance == 0 or checkCollision(x_new, x_nearest) then
        removeNode(x_new)
        continue
    end if
    X_near ← findNearestNodes(x_new)
    if x_new != x_goal then
        X_new ← updateParent(x_new, X_near)
        for x in X_new do
            rewire(x, X_near)
        end for
    else
        updateGoalParent(X_near)
    end if
end for
```

Table 3.1: *Average computation time and number of homotopy classes found for test runs of the modified RRT* algorithm.*

Headline	**Test 1**	**Test 2**
Average computation time [s]	9.38	17.19
Average number of homotopy classes found	5.61	15.08
Minimum number of homotopy classes found	3	4
Maximum number of homotopy classes found	10	32

3.1.1 Results

The modified RRT* algorithm is implemented in Python 3.7. In this section, the implementation is evaluated on two different scenarios with different obstacle placements and goal states. The cost function is chosen as the Euclidean distance and the search space is discrete. The maximum number of iterations is set to 2000 and the sampling bias towards the goal is set to $p = 0.05$. For each test scenario the algorithm has been executed 500 times. The resulting average computation times and number of homotopy classes found are shown in Table 3.1. Examples of trajectories from the two test scenarios are presented in Figures 3.1-3.2. Scatter plots of corresponding running times and number of homotopy classes found are presented in Figure 3.3 and Figure 3.4.

To further examine the relationship between running time and the number of homotopy classes each algorithm has been executed ten times. Each time a new homotopy class is found the running time is noted. The results are presented in Figure 3.5 and Figure 3.6.

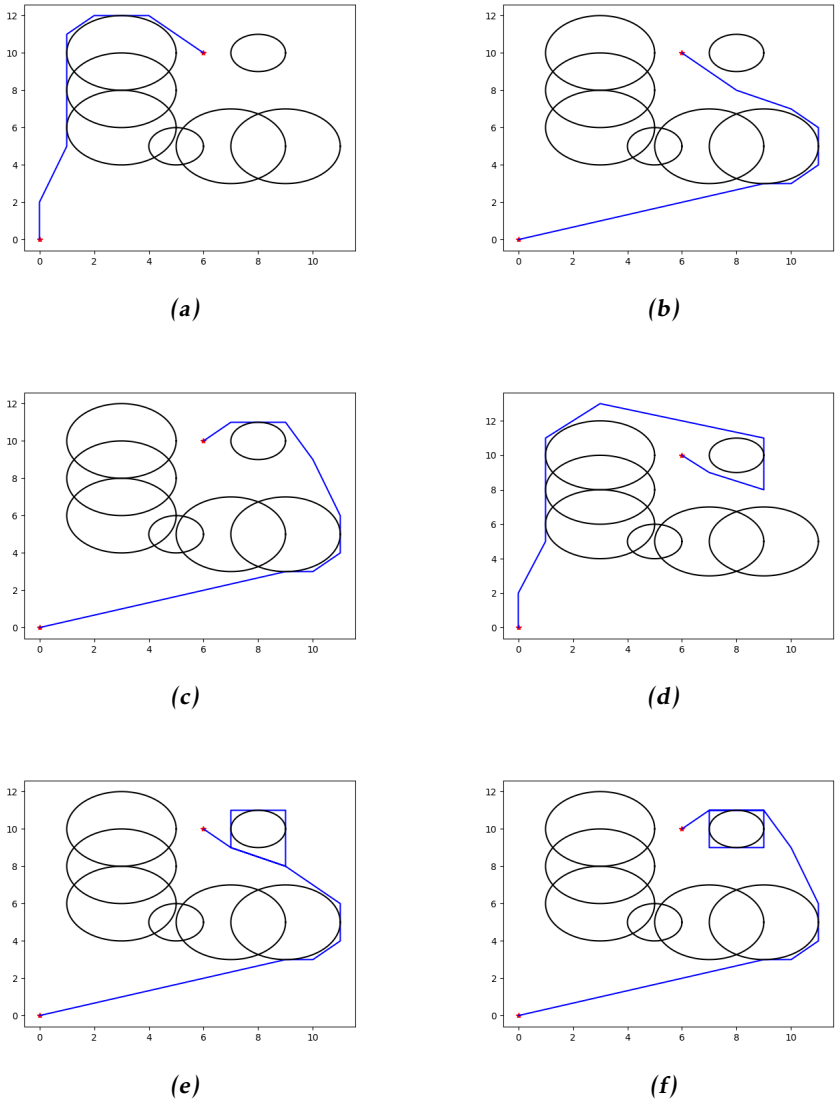


Figure 3.1: *Example of paths generated using the modified RRT* for the first test scenario. Each generated path belongs to a different homotopy class. The path costs are (a) 17.40, (b) 20.38, (c) 22.71, (d) 26.97, (e) 29.62 and (f) 30.71.*

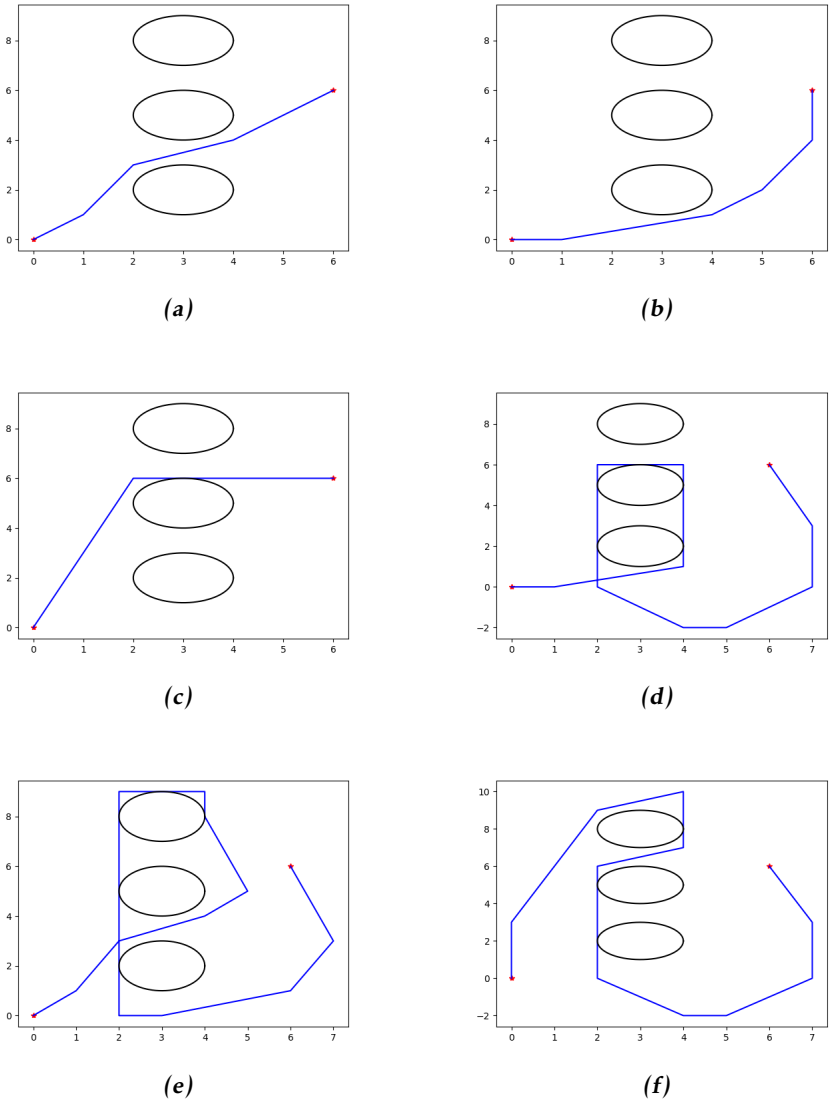


Figure 3.2: *Example of paths generated using the modified RRT* for the second test scenario. Each generated path belongs to a different homotopy class. Of the 14 found paths the three with the lowest and the three with the highest costs are shown. The path costs are (a) 8.71, (b) 9.81, (c) 10.32, (d) 29.98, (e) 32.02 and (f) 35.62.*

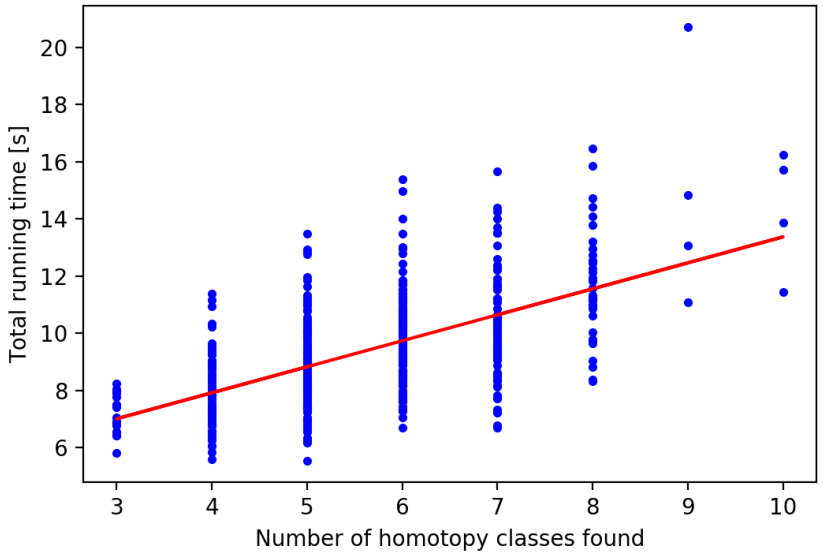


Figure 3.3: *Computation times and number of homotopy classes found for the first scenario. Trendline shown in red.*

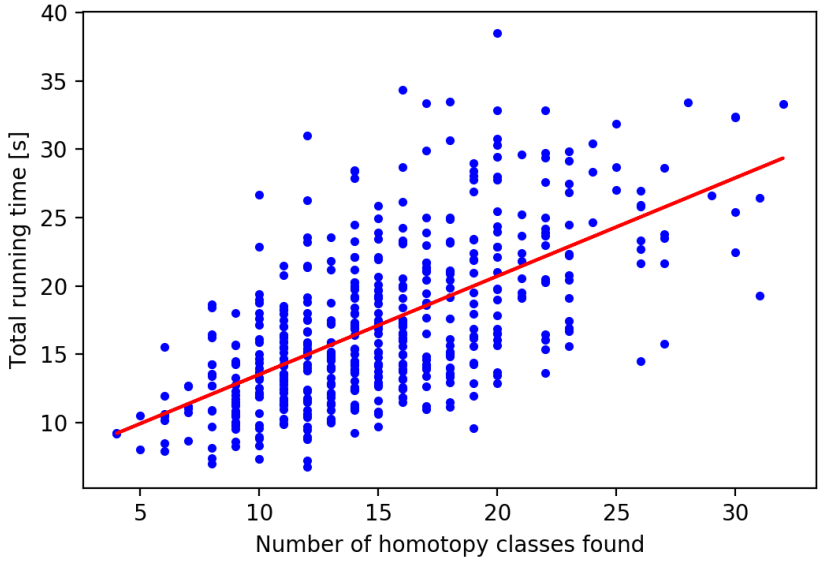


Figure 3.4: *Computation times and number of homotopy classes found for the second scenario. Trendline shown in red.*

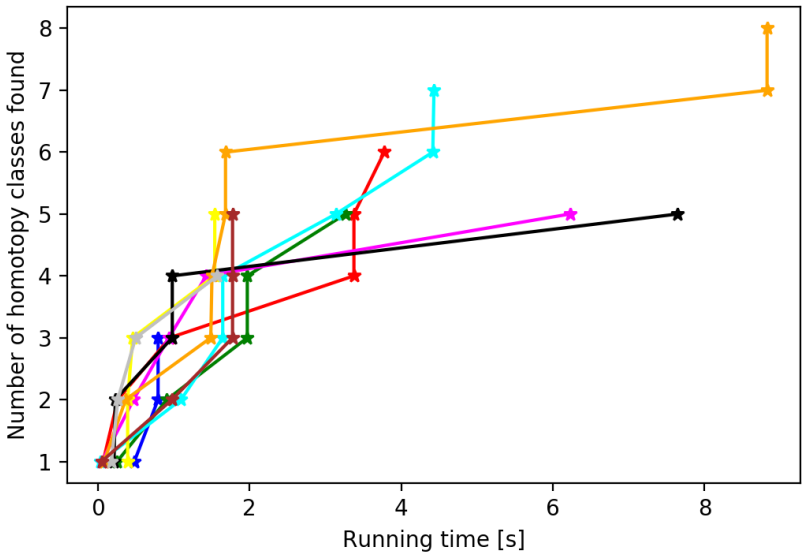


Figure 3.5: *The number of homotopy classes found as a function of the running time for the first scenario. Each of the ten executions is presented in a different color.*

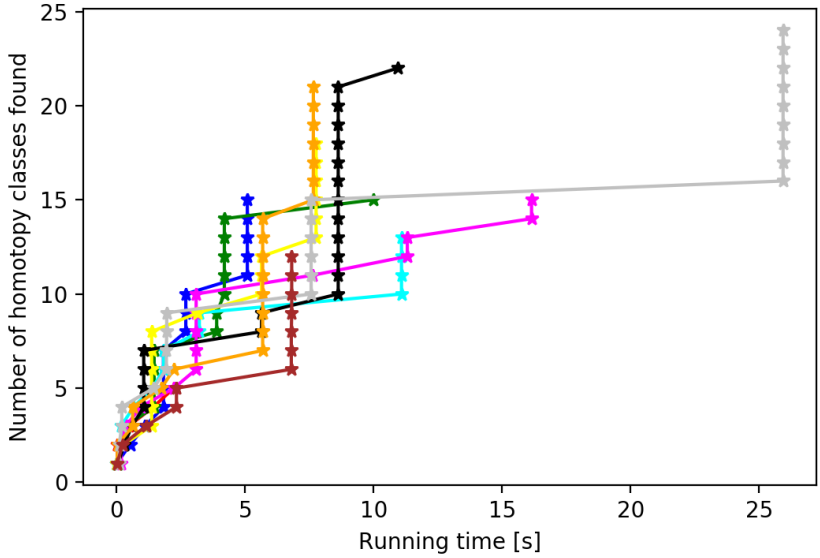


Figure 3.6: *The number of homotopy classes found as a function of the running time for the second scenario. Each of the ten executions is presented in a different color.*

3.2 Generation of initial trajectory

The resulting set of waypoints from the RRT* algorithm is processed and redundant waypoints are removed from the path. A waypoint is considered redundant if it lies on the straight line connecting the previous and the following waypoints.

One general approach to compute the initial trajectory is to use closed-loop simulation with a path-following controller that aims at following the waypoints computed by the modified RRT*. In the special case where the agent can be modelled as a car-like vehicle, the trajectory can be found by constructing Dubins paths between the waypoints.

The generated initial trajectory will not follow the path from the modified RRT* exactly since the modified RRT* does not consider dynamic constraints. It is therefore possible for the RRT* path to be collision-free while the generated trajectory is not. This issue can be solved by inflating obstacles for the RRT* step, and then deflating them later for the optimization step so as to have a collision-free initial trajectory. It would also be possible to modify the RRT* algorithm so as to consider dynamic constraints by modifying the steering function.

3.2.1 Dubins paths

The remaining waypoints after the waypoints returned from RRT* have been processed are connected using Dubins paths. In order to compute the Dubins paths the heading at each waypoint is determined. For all waypoints other than the initial and the final, the heading selected as the heading required to move from the current waypoint to the next in a straight line. In other words, if waypoint i is at position (x_i, y_i) and waypoint $i+1$ is at position (x_{i+1}, y_{i+1}), the heading ψ_i at waypoint i is selected as $\arctan2(y_{i+1} - y_i, x_{i+1} - x_i)$. The heading for the final waypoint is either selected as the same as the heading at the next-to-last waypoint, or assumed to have been given as part of the problem statement. The heading at the initial waypoint is assumed to have been given as part of the problem statement.

The points on the resulting Dubins paths are sampled at regular time intervals, for some sampling time δ_t. As in Section 2.4, the agent is assumed to be moving with constant speed v. The points on the Dubins paths should therefore be sampled at equal distance δ_d from each other, where $\delta_d = v\delta_t$. The control signals $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{l-1}$ required to follow the Dubins paths can be computed while constructing the path. Following the Dubins path requires only three different control inputs: the control input $\mathbf{u}_S$ that results in the agent moving forwards with contstant speed v, the control input $\mathbf{u}_R$ that results in the agent turning to the right in a circle with radius ρ and constant speed v, and the control input $\mathbf{u}_L$ that results in the agent turning to the left in a circle with radius ρ and constant speed v. The control input $\mathbf{u}_t$ at time t is then chosen as $\mathbf{u}_S$ if the robot is at a S-segment at time t, as $\mathbf{u}_R$ if the agent is at a R-segment at time t and as $\mathbf{u}_L$ if the agent is at a L-segment at time t.

3.2.2 Belief propagation

When the initial control inputs $\bar{\mathbf{u}}_0, \bar{\mathbf{u}}_1, \ldots, \bar{\mathbf{u}}_{l-1}$ have been computed it is possible to compute the beliefs $\bar{\mathbf{b}}_1, \ldots, \bar{\mathbf{b}}_l$ of the initial trajectory. This may also be done at the same time as computing the control inputs. The belief $\bar{\mathbf{b}}_0 = (\hat{\mathbf{x}}_0, \sqrt{\Sigma_0})$ at time $t = 0$ is assumed to be known. At each time step t the belief $\bar{\mathbf{b}}_t = (\hat{\mathbf{x}}_t, \sqrt{\Sigma_t})$ is updated according to (2.22)-(2.28).

The resulting trajectory $(\bar{\mathbf{b}}_0, \bar{\mathbf{u}}_0, \ldots, \bar{\mathbf{b}}_l$ can now be used as initial trajectory for iLQG or T-LQG. In iLQG, the initial trajectory is used as the first nominal trajectory. In T-LQG the initial trajectory is used as an initial guess for the NLP problem.

4

Motion planning under uncertainty

This chapter discusses the implementation of the two methods for motion planning under uncertainty.

- Section 4.1 discusses the implementation details for iLQG. The collision avoidance function is discussed in detail for a special case. The Hessians and Jacobians of the cost functions for some special cases are presented. Finally, a stopping criterion for the method is described.

- Section 4.2 discusses the implementation details for T-LQG. First, the OBF is presented for the special case of circular obstacles. Then, numerical issues in the optimization step are discussed and a solution to these issues is presented.

4.1 Iterative local optimization in belief space (iLQG)

This section discusses some specifics of the implementation of the iLQG method.

- Section 4.1.1 presents in detail how to compute the collision avoidance function for the special case when obstacles are circular. The problem of computing the shortest distance between a point and an ellipse is discussed.

- Section 4.1.2 describes how the Hessians and Jacobians of the cost functions can be computed analytically in special cases.

- Section 4.1.3 presents a stopping criterion for the iLQG algorithm.

4.1.1 Collision avoidance

It is not necessarily the case that all state variables are needed for collision detection and avoidance. In this thesis it is assumed that the state variables describing the spatial position of the agent are sufficient to detect collisions. Any other state variables such as velocity or orientation are assumed to be irrelevant for collision detection. In this thesis it is also assumed that the spatial position is two-dimensional, and hence that the dimension n referred to in Section 2.6.2 is $n = 2$.

Recall from Section 2.6.2 that a lower bound for the probability of not colliding could be expressed using the regularized gamma function $\gamma(s, x)$. When $s = 1$, the regularized gamma function simplifies to

$$\gamma(1, x) = \int_0^x e^{-t} dt = 1 - e^{-x}. \tag{4.1}$$

After combining (2.31) and (4.1), the collision avoidance function for $n = 2$ can be expressed as

$$f(\sigma(\mathbf{b}_t)) = -\ln[1 - e^{\sigma(\mathbf{b}_t)^2/2}]. \tag{4.2}$$

In this work, it is assumed that all obstacles can be modeled as circles. It is further assumed that there are K obstacles in total, all of which are contained in a two-dimensional subspace of the state space (i.e. $n = 2$). Let $\sigma_k(\mathbf{b}_t)$ denote the number of standard deviations it is possible to deviate from the mean before colliding with obstacle k, $k = 1, \ldots, K$. Then $\sigma(\mathbf{b}_t) = \min_{1 \leq k \leq K} \sigma_k(\mathbf{b}_t)$.

Let Z be the 2×2 submatrix of Σ_t that describes the covariance in the dimensions of the state space where the obstacles are found. The matrix $\sqrt{Z}$ describes an ellipse that contains all state configurations within one standard deviation [22]. The semi axes of the ellipse are parallel to the normalized eigenvectors v_1, v_2 of $\sqrt{Z}$, and the corresponding eigenvalues λ_1, λ_2 are equal to the lengths of the semi axes [5]. To compute $\sigma_k(\mathbf{b}_t)$, a series of transformations are applied to transform the ellipse into the unit circle centered at the origin. The following steps are performed:

1. The translation that maps the ellipse center to the origin is applied.
2. A transformation with transformation matrix $[v_1, v_2]^T$ is applied. The transformation is either a rotation or a composition of a rotation and a reflection. The ellipse is still centered at the origin and its semi axes are now aligned with the coordinate axes. The center of the obstacle circle is changed, but it remains a circle with the same radius as before.
3. The ellipse is transformed into the unit circle by scaling x-coordinates with $\frac{1}{\lambda_1}$ and y-coordinates with $\frac{1}{\lambda_2}$. This transforms the obstacle circle into an ellipse with semi axis lengths $\frac{r}{a}$ and $\frac{r}{b}$, where r is the circle radius. The semi axes are parallel to the coordinate axes.

The desired value $\sigma_k(\mathbf{b}_t)$ can now be computed as the distance from the origin to the obstacle ellipse. For an illustration of the steps above, see Figure 4.1.

Distance from a point to an ellipse

An ellipse, centered at (x_c, y_c) with semi axes lengths a, b and semi axes parallel to the $x-$ and $y-$ coordinate axis can be described by

$$(\frac{x - x_c}{a})^2 + (\frac{y - y_c}{b})^2 = 1. \tag{4.3}$$

The ellipse can be parameterized as $(x_c + a\cos\theta, y_c + b\sin\theta)$. Let (x_p, y_p) be a point from which the shortest distance to the ellipse is to be computed. The nearest point on the ellipse is the point for which the function

$$f(\theta) = \|(x_c - x_p + a\cos\theta, y_c - y_p + b\sin\theta)\|_2^2 \tag{4.4}$$

is minimized. A necessary (but not sufficient) condition for θ to be the minimizer is that $f'(\theta) = 0$. Standard calculations give that:

$$f(\theta) = (x_c - x_p + a\cos\theta)^2 + (y_c - y_p + b\sin\theta)^2 \tag{4.5}$$

$$f'(\theta) = (b^2 - a^2)\sin\theta\cos\theta - a(x_c - x_p)\sin\theta + b(y_c - y_p)\cos\theta. \tag{4.6}$$

In order to solve $f'(\theta) = 0$, Newtons method can be used with initial guess $\theta_0 = \arctan2(a(y_p - y_c), b(x_p - x_c))$ which in practice often converges within a few iterations [12]. This initial guess is only good as long as (x_p, y_p) does not lie inside the ellipse. That constraint can easily be checked by computing $k = (\frac{x_p - x_c}{a})^2 + (\frac{y_p - y_c}{b})^2$. If $k \leq 1$ then the point is either inside or on the ellipse. In this case it is not possible to deviate from the mean without risking collision, hence $\sigma_k(\mathbf{b}_t) = 0$. This causes $f(\sigma_k(\mathbf{b}_t))$ to tend to infinity which makes it numerically unstable to compute the Hessian and Jacobian of $f(\sigma_k(\mathbf{b}_t))$, which is discussed in Section 4.1.2.

4.1.2 Hessians and Jacobians of cost functions

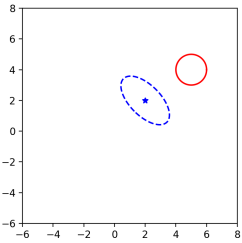
The iLQG requires the computation of Hessians and Jacobians of the cost functions in (2.32)-(2.33), which in some special cases can be done analytically.

According to [23], the Hessian and Jacobian of $f(\sigma(\bar{\mathbf{b}}_t))$ can be approximated as $a\mathbf{a}\mathbf{a}^T$ and $b\mathbf{a}$ respectively, where $a = \frac{\partial^2 f}{\partial \sigma^2}(\sigma(\bar{\mathbf{b}}_t))$, $b = \frac{\partial f}{\partial \sigma}(\sigma(\bar{\mathbf{b}}_t))$ and $\mathbf{a}^T = \frac{\partial \sigma}{\partial \mathbf{b}}(\bar{\mathbf{b}}_t)$. Differentiating equation 4.2 twice gives

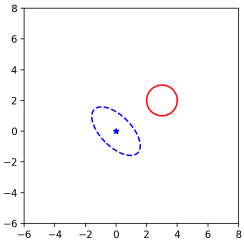
$$b = \frac{\sigma(\bar{\mathbf{b}}_t)}{1 - e^{\sigma(\bar{\mathbf{b}}_t)^2/2}} \tag{4.7}$$

$$a = \frac{(\sigma(\bar{\mathbf{b}}_t)^2 - 1)e^{\sigma(\bar{\mathbf{b}}_t)^2/2} + 1}{(1 - e^{\sigma(\bar{\mathbf{b}}_t)^2/2})^2}. \tag{4.8}$$

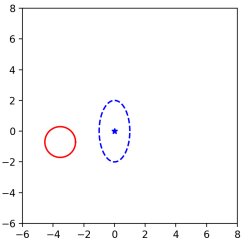
These expressions are well defined for all $\sigma(\bar{\mathbf{b}}_t) > 0$. To avoid singularities for $\sigma(\bar{\mathbf{b}}_t) = 0$ one approach is to choose $\sigma_{min} > 0$ and let $\sigma(\bar{\mathbf{b}}_t) = \sigma_{min}$ if $\sigma(\bar{\mathbf{b}}_t) < \sigma_{min}$.



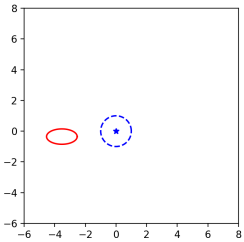
(a) An example of an uncertainty ellipse (blue) centered at (2, 2) and a circular obstacle (red) centered at (5, 4).



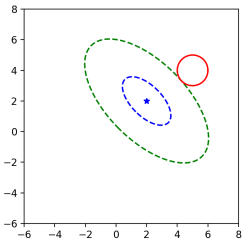
(b) Step 1. The ellipse and circle are translated so that the ellipse is centered at the origin.



(c) Step 2. The ellipse and circle are rotated (or rotated and reflected) so that the semi axes of the ellipse are parallel to the coordinate axes.



(d) Step 3. The ellipse and circle are scaled along the x- and y-axes so that the ellipse is mapped to the unit circle. The circle is mapped to an ellipse. $\sigma(\mathbf{b}_t)$ can now be computed as the distance from the origin to the transformed obstacle (red).



(e) The original uncertainty ellipse (blue) and obstacle (red) shown together with the uncertainty ellipse scaled by $\sigma(\mathbf{b}_t)$ (green). Note that the scaled uncertainty ellipse touches the obstacle.

Figure 4.1: *Example of the translation, rotation and scaling performed to transform the uncertainty ellipse into the unit circle centered at the origin.*

Standard matrix calculations give that:

$$\frac{\partial}{\partial \mathbf{u}} \mathbf{u}^T R_t \mathbf{u} = (R_t + R_t^T)\mathbf{u} \tag{4.9}$$

$$\frac{\partial^2}{\partial \mathbf{u} \partial \mathbf{u}} \mathbf{u}^T R_t \mathbf{u} = R_t + R_t^T \tag{4.10}$$

$$\frac{\partial}{\partial \hat{\mathbf{x}}_l} (\hat{\mathbf{x}}_l - \mathbf{x}_{goal})^T Q_l (\hat{\mathbf{x}}_l - \mathbf{x}_{goal}) = (Q_l + Q_l^T)(\hat{\mathbf{x}}_l - \mathbf{x}_{goal}) \tag{4.11}$$

$$\frac{\partial^2}{\partial \hat{\mathbf{x}}_l \partial \hat{\mathbf{x}}_l} (\hat{\mathbf{x}}_l - \mathbf{x}_{goal})^T Q_l (\hat{\mathbf{x}}_l - \mathbf{x}_{goal}) = Q_l + Q_l^T. \tag{4.12}$$

Suppose a function of the form $c(X) = \text{tr}(X^T Q X)$, where $Q = \lambda I$ for some $\lambda \in \mathbb{R}$ and X is symmetric. Then

$$c(X) = \text{tr}(X^T Q X) = \lambda \, \text{tr}(X^T X) = \lambda \sum_{i=1}^{n} (X^T X)_{ii} = \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} x_{ji} = \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^2, \tag{4.13}$$

where $x_{ij} = (X)_{ij}$. Due to the symmetry of X the expression can be rewritten as

$$c(X) = \text{tr}(X^T Q X) = \lambda \Big(\sum_{i=1}^{n} x_{ii}^2 + \sum_{i=2}^{n} \sum_{j=1}^{i-1} 2 x_{ij}^2 \Big). \tag{4.14}$$

It follows that $\frac{\partial c}{\partial x_{ii}}(X) = 2\lambda x_{ii}$ and $\frac{\partial c}{\partial x_{ij}}(X) = 4\lambda x_{ij}$, $i \neq j$. It follows also that

$$\frac{\partial^2}{\partial x_{ij} \partial x_{kl}} = \begin{cases} 2\lambda & , i = j = k = l \\ 4\lambda & , i = k, j = l, i \neq j \\ 0 & , \text{otherwise.} \end{cases} \tag{4.15}$$

4.1.3 Stopping criterion

The iLQG method iteratively computes a control policy which is used to compute the next nominal trajectory, as described in Section 2.6.2. The algorithm is to continue until convergence, i.e. until some stopping criterion is met. In this thesis, the algorithm is stopped either when ε in (2.30) falls below a preset value, or when the improvement in objective function between two subsequent iterations is smaller than a threshold.

4.2 T-LQG

The T-LQG method, as described in Section 2.6.3, separates the motion planning into computation of an optimized trajectory and computation of a control policy. Finding the optimized trajectory requires the solution of an NLP problem. In this thesis, the NLP is solved using the CasADi framework (version 3.5.1) [1]

combined with the NLP solver WORHP (version 1.13) [18]. Even though auxiliary variables for $\hat{\mathbf{x}}_0, \ldots, \hat{\mathbf{x}}_l$ and $\Sigma_0, \ldots, \Sigma_l$ are added in the implementation, the objective function is considered to be a function of the control inputs $\mathbf{u}_0, \ldots, \mathbf{u}_{l-1}$ only. The resulting series of control inputs $(\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{l-1})$ and the initial belief $\mathbf{b}_0$ are used to compute the series of beliefs $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_l$.

4.2.1 Collision avoidance

As for the iLQG in Section 4.1.1, it is supposed that there are K obstacles in total, all circular, and that obstacle k has radius r_k and center c_k. The OBF is then:

$$\Phi(x) = \sum_{i=1}^{K} [M_1 \exp(-r_i \|x - c_i\|_2^{2q}) + M_2 \sum_{\theta=0:\frac{1}{m}:1} (\|x - (\theta\zeta^{i,1} + (1-\theta)\zeta^{i,2})\|_2^{-2} + \|x - (\theta\xi^{i,1} + (1-\theta)\xi^{i,2})\|_2^{-2})]. \tag{4.16}$$

where $\zeta^{i,1}, \zeta^{i,2}$ and $\xi^{i,1}, \xi^{i,2}$ are the endpoints of the circle semi-axes, and M_1, M_2, m, q are (positive) parameters defined by the user.

The OBF cost function of (2.38) can be approximated as

$$c_{obf}(\mathbf{x}_t) = \int_{x_{t1}}^{x_{t2}} \Phi(x')dx' \approx \Phi(x_{t2})\|x_{t2} - x_{t1}\|_2. \tag{4.17}$$

Inserting this approximation into (2.39) results in the cost function

$$\sum_{t=1}^{l} [\mathrm{tr}(W_t \Sigma_t W_t^T) + \mathbf{u}_{t-1}^T W_t^u \mathbf{u}_{t-1} + \Phi(x_t)\|x_t - x_{t-1}\|_2]. \tag{4.18}$$

4.2.2 Optimization step

When the trajectory optimization problem is solved in CasADi and WORHP, optimization variables for $\mathbf{u}_t, \hat{\mathbf{x}}_t, \Sigma_t$ are all added. Due to numerical issues it sometimes happens that for some t the covariance matrix Σ_t is not positive semi-definite. This issue is solved by considering all other optimization variables to be functions of $\mathbf{u}_t$. The initial belief $\mathbf{b}_0$ and the resulting control inputs $\mathbf{u}_0, \ldots, \mathbf{u}_{l-1}$ can then be used to compute the resulting trajectory by forward propagation as described in Section 3.2.2.

5
Simulation study

This chapter describes the evaluations performed in the simulation study as well as presents the results.

- Section 5.1 describes the evaluations and simulations performed in order to compare the two implemented methods. The three different scenarios that are used are presented.
- Section 5.2 defines the model of the agent used in the evaluations.
- Section 5.3 defined the cost functions used in the evaluations and presents the choices of cost function parameters.
- Sections 5.4-5.6 present the results of the evaluations for the three scenarios.

5.1 Method

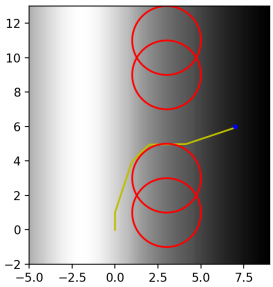
To evaluate the iLQG and T-LQG methods three different scenarios are constructed with different obstacle placement, different goal states and different uncertainty distributions. The modified RRT* algorithm is used initially on each scenario to find paths from which initial trajectories are computed using Dubins paths. The three scenarios and their generated Dubins paths are shown in Figure 5.1. The first scenario is a simple test where only one path is generated. In the second test scenario the start point is in an area with higher measurement uncertainties and the goal is in an area with lower measurement uncertainties. In this scenario, three different paths are generated using the modified RRT* algorithm. The first path (the shortest) leads the agent through a narrow passage between two obstacles. The second and third paths (that are longer) both avoid the narrow passage,

and instead arrives to the goal by rounding one of the obstacles. In the third scenario, the initial point and the goal point have the same x-coordinates. However, the agent can not travel in a straight line due to an obstacle. In this scenario, two paths are generated using the modified RRT* algorithm. The first path (the shortest) leads the agent slightly to the left in order to avoid the obstacle, passing through an area with higher measurement uncertainties. The second path (the longest) rounds the obstacle on the right hand side, passing through an area with lower measurement uncertainties.

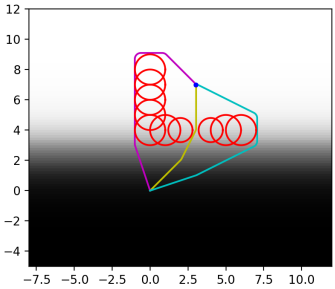
An initial trajectory is generated for each Dubins path in the three scenarios. The iLQG and T-LQG methods have both been applied to each initial trajectory and the resulting cost function values are noted. In order to make a comparison between the two methods possible, both cost functions are evaluated for the resulting optimized trajectories, as well as for the initial trajectories.

Both methods are applied three times to each initial trajectory and the average computation time for the trajectory optimization and policy computation is reported. For T-LQG, the computation times when using the T-LQG method control policy and when using the iLQG method control are both reported.

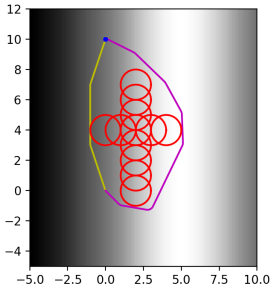
To further evaluate the two methods, 100 simulations of the agent following the computed trajectories have been performed for the two different methods. For T-LQG, simulations are also performed using a control policy and state estimator computed according to the iLQG method. Each state along the resulting simulated trajectories is checked for collision by checking if it lies inside any obstacle. If any state lies inside an obstacle, it is considered a collision. For simulated trajectories that did not result in collision, the distance between the true final position and the desired final position is reported. In all scenarios the initial (true) position is generated randomly as $\mathbf{b}_0 + w$, where $w \sim \mathcal{N}(\mathbf{0}, \text{diag}(0.5, 0.5, 0.001, 0.035))$.



(a) *First scenario. The generated path is shown in yellow.*



(b) *Second scenario. The three generated paths are shown in yellow (first path), purple (second path) and cyan (third path).*



(c) *Third scenario. The two generated paths are shown in yellow (first path) and purple (second path).*

Figure 5.1: *The three scenarios and their generated paths. Red circles represent obstacles. Measurement uncertainties are greater in darker areas. The goal is marked with a blue dot in each scenario.*

5.2 Agent model

The agent in the simulation is a fixed-wing aircraft flying at constant height. The state of the aircraft is $\mathbf{x} = (x, y, \theta, v)$ where x, y describe the position of the aircraft, θ is the heading and $v \geq 0$ is the speed of the aircraft. The control input is $\mathbf{u} = (a, \phi)$, where a is the acceleration of the aircraft and ϕ is the steering angle. The dynamical model of the aircraft is

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{m}_t) = \begin{pmatrix} x_t + \Delta_t v_t \cos\theta_t \\ y_t + \Delta_t v_t \sin\theta_t \\ \theta_t + \Delta_t v_t \tan(\phi)/d \\ v_t + \Delta_t a_t \end{pmatrix} + M\mathbf{m}_t \tag{5.1}$$

where Δ_t is the length of a time step, d is the aircraft length and M is a matrix that scales the motion noise $\mathbf{m}_t$. This is the same as the model of a simple car found in [9], only with noise added. Although the length of the agent is included in the model, the agent is treated as a single point for the collision detection.

In all three scenarios the M-value used is

$$M = \begin{pmatrix} 0.007 & 0 & 0 & 0 \\ 0 & 0.007 & 0 & 0 \\ 0 & 0 & 5 * 10^{-7} & 0 \\ 0 & 0 & 0 & 0.001 \end{pmatrix}. \tag{5.2}$$

The observation model is

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{n}_t) = \mathbf{x}_t + N(\mathbf{x}_t)\mathbf{n}_t \tag{5.3}$$

where $N(\mathbf{x}_t)$ is a matrix that scales the sensing noise $\mathbf{n}_t$ as a function of the state $\mathbf{x}_t$.

5.3 Cost functions

The iLQG cost function is defined by (2.1) and (2.32)-(2.33), where the term $f(\sigma(\mathbf{b}_t))$ is computed according to the description in Section 4.1.1. In all three scenarios the weight matrices of (2.32)-(2.33) are selected as $Q_l = 10lI$ and $Q_t = R_t = I$, where l is the time horizon. The term $f(\sigma(\mathbf{b}_t))$ in (2.33) is scaled by the factor 0.2 in all three scenarios

The T-LQG cost function is defined by (4.16) and (4.18). The weight matrices in (4.18) are selected as $W_t = W_t^u = I$ in the first two scenarios. The third scenario uses $W_t = I$, $W_t^u = 5I$. The parameters q and m in (4.16) are selected as $q = 1$ and $m = 10$ for all three scenarios. The parameter M_1 in (4.16) is selected as $M_1 = 1$ for the first two scenarios and as $M_1 = 5$ for the third scenario. The parameter M_2 in (4.16) is selected as $M_2 = 0.015$ in all three scenarios.

Table 5.1: *Cost function values for the initial and optimized trajectories for the first scenario.*

Trajectory	iLQG cost function	T-LQG cost function
Initial trajectory	193.03	20.49
iLQG trajectory	42.85	9.38
T-LQG trajectory	44.38	8.39

Table 5.2: *Average computation times for the first scenario.*

Trajectory	Computation time [s]
iLQG trajectory	285.07
T-LQG trajectory (T-LQG control policy)	9.98
T-LQG trajectory (iLQG control policy)	12.44

5.4 Scenario 1 - initial evaluation

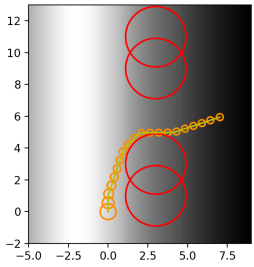
The first scenario, presented in Figure 5.1a, is a simple scenario where only one trajectory is evaluated. The trajectory starts at $(0, 0)$ and the goal is at $(7, 6)$. There are two obstacles, both centered at $x = 3$ and centered at different $y-$coordinates. The sensing noise is scaled along the x-axis according to

$$N_t(\mathbf{x_t}) = \begin{pmatrix} 0.5n(x_t) & 0 & 0 & 0 \\ 0 & 0.5n(x_t) & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.035 \end{pmatrix} \tag{5.4}$$

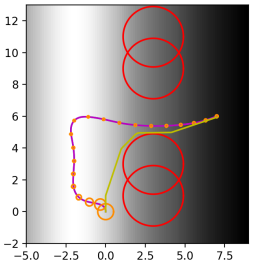
where $n(x) = 1 + 0.5(x + 2)^2$ and x_t is the x-coordinate at time t. This means that the measurement uncertainty for (5.3) is smaller when approaching $x = -2$ and increases with the quadratic distance to $x = -2$.

The initial trajectory along with the resulting optimized trajectories for the two methods are shown in Figure 5.2. The initial trajectory passes between the two obstacles, following the lower obstacle closely. The two optimized trajectories both initially lead the agent to the left until reaching the area with lowest measurement uncertainty. They then lead the agent up before turning right and heading towards the goal. The T-LQG trajectory passes between the two obstacles at about equal distance to both, whereas the iLQG trajectory is closer to the lower obstacle. Both optimized trajectories show reduced uncertainty compared to the initial trajectory.

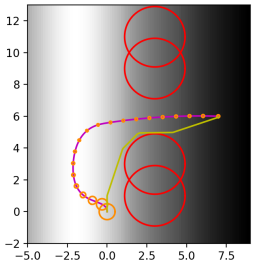
The resulting cost function values for the trajectories are presented in Table 5.1. The computation times are presented in Table 5.2. The initial and optimized trajectories are shown in Figure 5.2. The results from simulations of the agent following the computed trajectories are shown in Figure 5.3.



(a) *Initial trajectory.*



(b) *iLQG trajectory.*



(c) *T-LQG trajectory.*

Figure 5.2: *Initial (yellow) and optimized (purple) trajectories for the first scenario. Uncertainty ellipses are shown in orange. Red circles represent obstacles. Measurement uncertainties are higher in darker areas.*

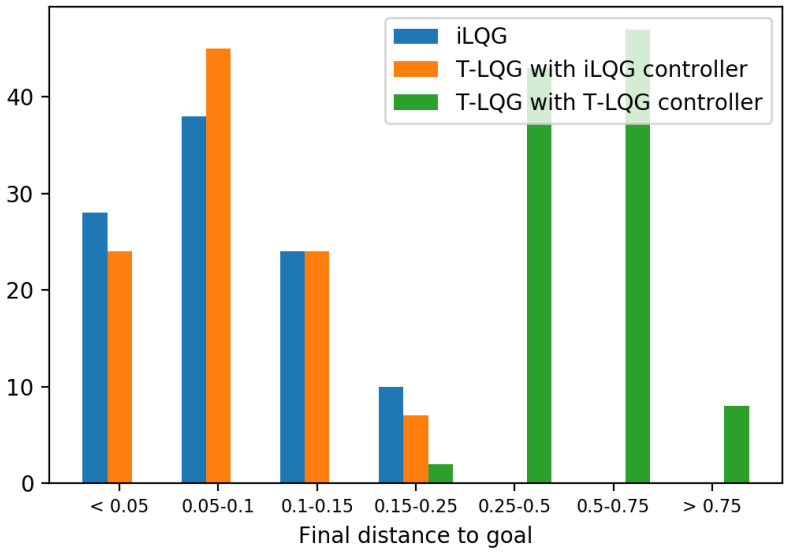


Figure 5.3: *Distribution of distance between final position and goal position over 100 simulations for the first scenario.*

5.5 Scenario 2 - narrow passage

In the second scenario, as seen in Figure 5.1b, the starting point is $(0, 0)$ and the goal point is $(3, 7)$. The goal is surrounded by obstacles to the left and below, with only a narrow passage below. There are three different paths from which three different initial trajectories are computed. The first path takes the shortest way, leading the agent through the narrow passage. The other paths are longer, but avoid the passage. The sensing noise is scaled along the y-axis according to

$$N_t(\mathbf{x_t}) = \begin{pmatrix} 0.5n(y_t) & 0 & 0 & 0 \\ 0 & 0.5n(y_t) & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.035 \end{pmatrix} \tag{5.5}$$

where $n(y) = 1 + \frac{1}{1+e^{y-3}}$ and y_t is the $y-$coordinate at time t. This means that the measurement uncertainty varies as a sigmoid function of the $y-$coordinate. The higher y-coordinate, the lower measurement uncertainty.

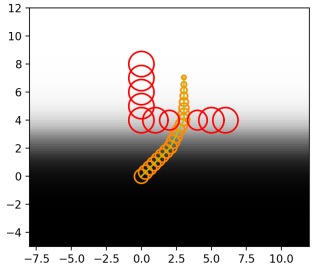
The initial and optimized trajectories for the three paths are shown in Figures 5.4-5.6. The resulting cost function values are presented in Table 5.3. The running times are presented in Table 5.4. The results from the simulations are shown in Figure 5.7.

Table 5.3: *Cost function values for the initial and optimized trajectories for the second scenario.*

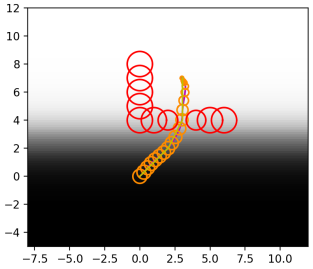
Trajectory	**iLQG cost function**	**T-LQG cost function**
Initial trajectory, first path	166.84	49.36
iLQG trajectory, first path	138.19	56.39
T-LQG trajectory, first path	114.57	42.94
Initial trajectory, second path	156.33	55.30
iLQG trajectory, second path	57.35	34.49
T-LQG trajectory, second path	52.17	31.32
Initial trajectory, third path	186.49	73.04
iLQG trajectory, third path	155.78	72.62
T-LQG trajectory, third path	63.57	41.78

Table 5.4: *Average computation times for the second scenario.*

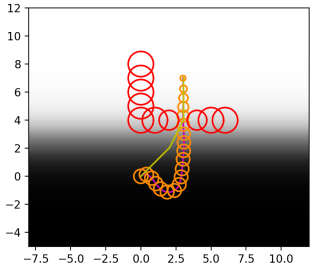
Trajectory	**Computation time [s]**
iLQG trajectory, first path	69.24
T-LQG trajectory (T-LQG control policy), first path	57.50
T-LQG trajectory (iLQG control policy), first path	58.91
iLQG trajectory, second path	829.23
T-LQG trajectory (T-LQG control policy), first path	23.67
T-LQG trajectory (iLQG control policy), second path	27.78
iLQG trajectory, third path	115.60
T-LQG trajectory (T-LQG control policy), third path	40.51
T-LQG trajectory (iLQG control policy), third path	45.09



(a) *Initial trajectory, first path.*

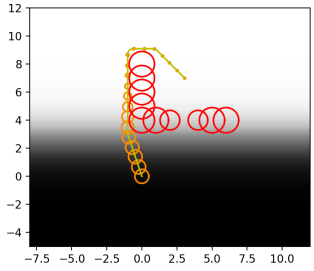


(b) *iLQG trajectory, first path.*

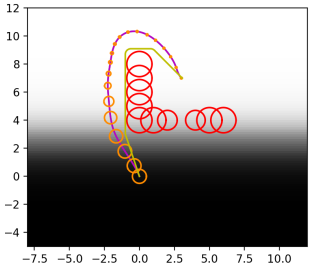


(c) *T-LQG trajectory, first path.*

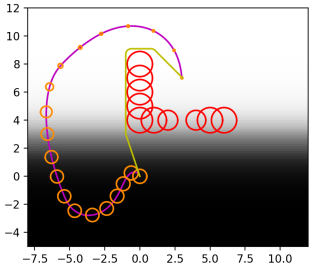
Figure 5.4: *Initial (yellow) and optimized (purple) trajectories for the first path of the second scenario. Uncertainty ellipses are shown in orange. Red circles represent obstacles. Measurement uncertainties are greater in darker areas.*



(a) *Initial trajectory, second path.*

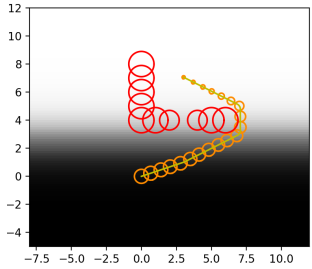


(b) *iLQG trajectory, second path.*

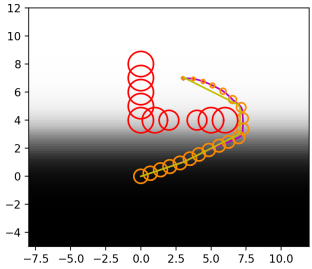


(c) *T-LQG trajectory, second path.*

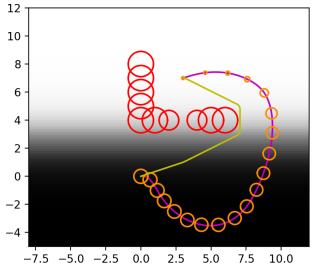
Figure 5.5: *Initial (yellow) and optimized (purple) trajectories for the second path of the second scenario. Uncertainty ellipses are shown in orange. Red circles represent obstacles. Measurement uncertainties are greater in darker areas.*



(a) *Initial trajectory, third path.*

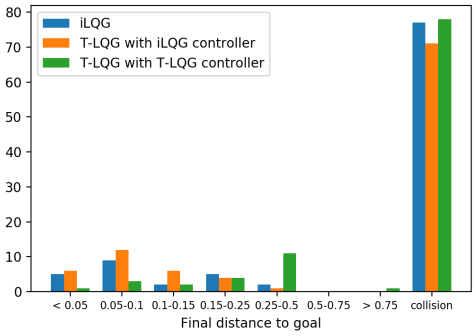


(b) *iLQG trajectory, third path.*

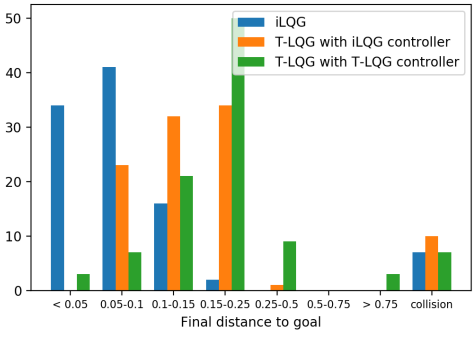


(c) *T-LQG trajectory, third path.*

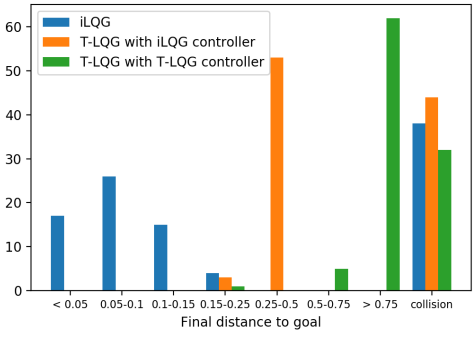
Figure 5.6: *Initial (yellow) and optimized (purple) trajectories for the third path of the second scenario. Uncertainty ellipses are shown in orange. Red circles represent obstacles. Measurement uncertainties are greater in darker areas.*



(a) First path.



(b) Second path.



(c) Third path.

Figure 5.7: *Distribution of distance between final position and goal position over 100 simulations for the second scenario. Collision means that the simulation ended in collision and did not reach a final position.*

5.6 Scenario 3 - multiple paths

In the third scenario, presented in Figure 5.1c, two different paths are evaluated. The starting point is $(0, 0)$ and the goal point is $(0, 10)$. There is only one obstacle. The first path makes a slight turn to the left in order to pass the obstacle, whereas the second path rounds the obstacle on the right-hand side which requires an initial turn downwards. The sensing noise is scaled along the x-axis according to

$$N_t(\mathbf{x_t}) = \begin{pmatrix} 0.5n(x_t) & 0 & 0 & 0 \\ 0 & 0.5n(x_t) & 0 & 0 \\ 0 & 0 & 0.001 & 0 \\ 0 & 0 & 0 & 0.035 \end{pmatrix} \tag{5.6}$$

where $n(x) = 1 + 0.3(x - 5)^2$ and x_t is the x-coordinate at time t. This means that the measurement uncertainty is smaller when approaching $x = 5$. The first, shorter path that passes the obstacle on the left side therefore passes through areas with higher measurement uncertainty than the second and longer path does.

The initial and optimized trajectories are shown in Figure 5.8. The resulting cost function values are presented in Table 5.5. The running times are presented in Table 5.6. The results from the simulations are shown in Figure 5.9.

5.6.1 Modified cost function for T-LQG

In a modification of the third test scenario, the cost function for T-LQG as described in Section 5.3 is slightly modified. Instead of using the same weight matrix W_t^x for all time steps t, a different weight matrix W_l^x is used for the final time step $t = l$ similarly to the cost function for iLQG. (Recall from Section 2.6.3 that $W_t^x = W_t^T W_t$.) The weight matrices are selected as $W_t^x = I$ and $W_l^x = 10lI$.

The resulting T-LQG trajectory for the first path is presented in Figure 5.10. The resulting cost function values are presented in Table 5.7. The average computation times over three executions are presented in Table 5.8. The results from 100 trajectory tracking simulations with a T-LQG controller and an iLQG controller are seen in Figure 5.11.

The modified T-LQG cost function has been evaluated on the second path of the third scenario as well, but no optimized trajectory could be computed. WORHP failed to return a solution to the trajectory optimization problem, returning the message *Unsuccessful termination: Converged to local point of infeasibility.* after 138 iterations and 123 seconds.

Table 5.5: *Cost function values for the initial and optimized trajectories for the third scenario.*

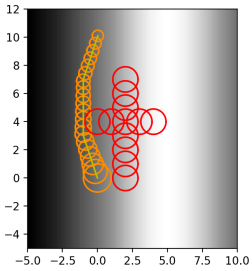
Trajectory	iLQG cost function	T-LQG cost function
Initial trajectory, first path	682.47	107.03
iLQG trajectory, first path	167.33	118.28
T-LQG trajectory, first path	604.30	95.03
Initial trajectory, second path	145.01	69.26
iLQG trajectory, second path	105.09	62.01
T-LQG trajectory, second path	115.61	45.63

Table 5.6: *Average computation times for the third scenario.*

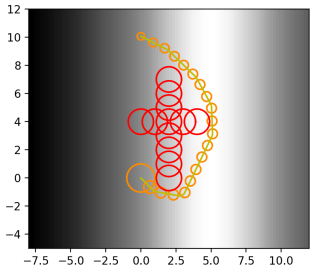
Trajectory	Computation time [s]
iLQG trajectory, first path	456.89
T-LQG trajectory (T-LQG control policy), first path	55.66
T-LQG trajectory (iLQG control policy), first path	54.89
iLQG trajectory, second path	508.73
T-LQG trajectory (T-LQG control policy), second path	27.32
T-LQG trajectory (iLQG control policy), second path	34.83

Table 5.7: *Cost function values for the initial and optimized trajectories for the first path of the third scenario with modified T-LQG cost function.*

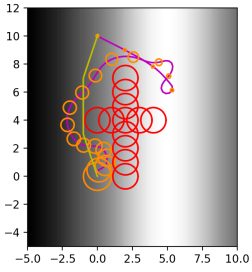
Trajectory	iLQG cost function	T-LQG cost function
Initial trajectory, first path	682.47	584.34
iLQG trajectory, first path	167.33	146.96
T-LQG trajectory, first path	141.37	112.61



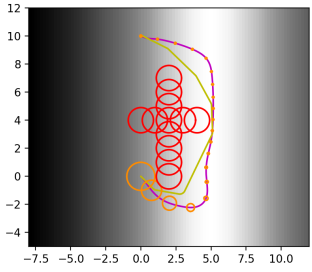
(a) Initial trajectory, first path.



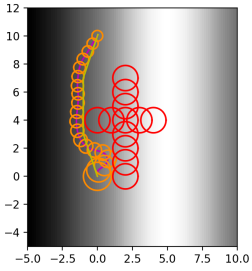
(b) Initial trajectory, second path.



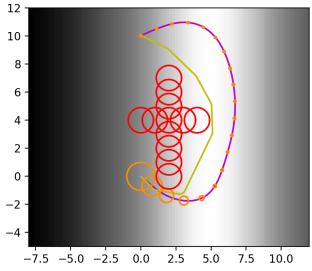
(c) iLQG trajectory, first path.



(d) iLQG trajectory, second path.

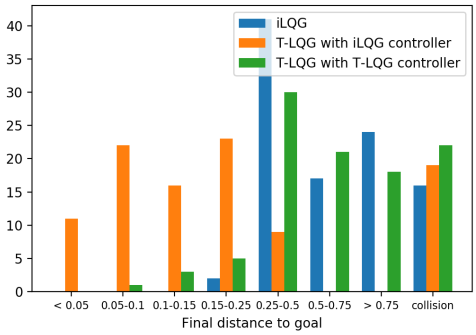


(e) T-LQG trajectory, first path.

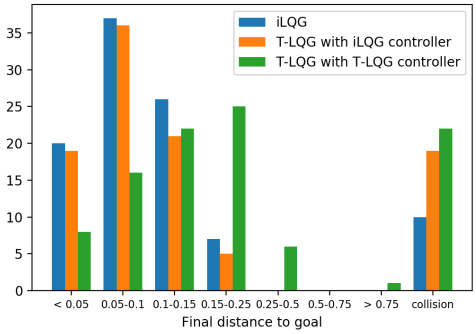


(f) T-LQG trajectory, second path.

Figure 5.8: *Initial (yellow) and optimized (purple) trajectories for the third scenario. Uncertainty ellipses are shown in orange. Red circles represent obstacles. Measurement uncertainties are greater in darker areas.*



(a) First path.



(b) Second path.

Figure 5.9: *Distribution of distance between final position and goal position over 100 simulations for the third scenario. Collision means that the simulation ended in collision and did not reach a final position.*

Table 5.8: *Average computation times for the first path of the third scenario with modified T-LQG cost function.*

Trajectory	**Computation time [s]**
T-LQG trajectory (T-LQG control policy), first path	52.29
T-LQG trajectory (iLQG control policy), first path	55.92

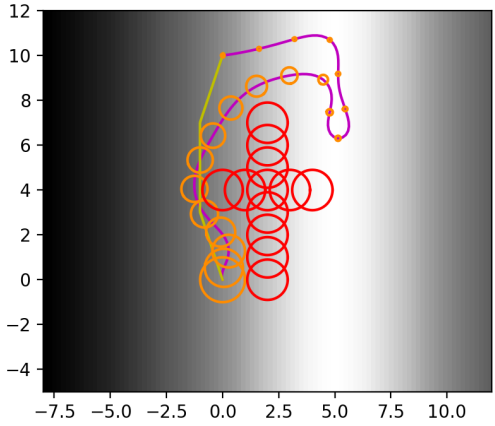


Figure 5.10: *Initial (yellow) and optimized (purple) trajectories for the first path of the third scenario with modified T-LQG cost function. Uncertainty ellipses are shown in orange. Red circles represent obstacles. Measurement uncertainties are greater in darker areas.*

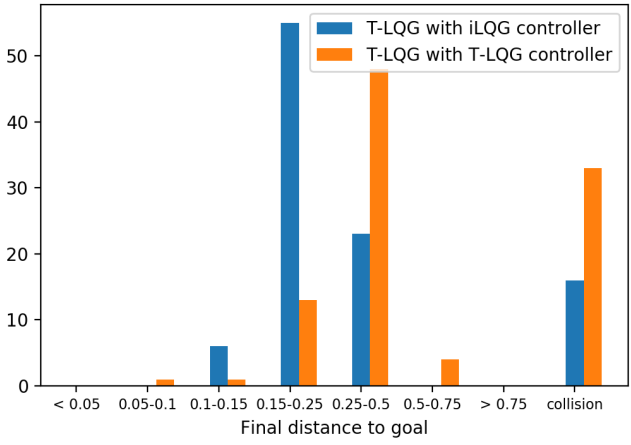


Figure 5.11: *Distribution of distance between final position and goal position over 100 simulations for the first path of the third scenario with modified T-LQG cost function.*

6
Discussion

This chapter contains a discussion of the results presented in Section 3.1.1 and Chapter 5.

6.1 Modified RRT* with homotopy

As seen in Table 3.1 the number of paths found by the modified RRT* algorithm varies greatly, both between different setups and between different runs on the same setup. This is due to the fact that RRT* is a stochastic algorithm rather than a deterministic. In general, it is reasonable to assume that only a few homotopy classes will be interesting in practice (a path that circles an obstacle multiple times will not be very time or energy efficient). It is also reasonable to assume that the interesting paths will be among the shorter paths, which are more likely to be found faster by the algorithm. A possible improvement of the algorithm could therefore be to limit the number of homotopy classes that the algorithm is allowed to find. Since the results in Figures 3.3-3.4 seem to indicate a correlation between longer running time and a higher number of homotopy classes, a limited amount of homotopy classes could serve to make the algorithm more efficient. This limitation will also enable the algorithm to focus on optimizing the paths in the existing homotopy classes instead of searching for new ones. However, the algorithm is not deterministic and there is no guarantee that homotopy classes will be found in an order that corresponds to the length of their shortest path. Hence, a reasonable trade-off is to select the number of homotopy classes somewhat higher than the number of paths that are desired in order to increase the probability that the shortest paths are all found.

From Figure 3.5 and 3.6 it can be seen that the time required to find a new homo-

topy class varies between executions. In general, it appears that new homotopy classes are found faster in the beginning, and that later in the algorithm there are longer stretches of time where no new homotopy classes are found. Another pattern that can be seen in Figure 3.6, is that often several new homotopy classes seem to be found at the same time. The explanation for this is that when the updateGoalParent function, as described in Section 3.1, is called there can be several new nodes that have not been tested as hypothetical parent nodes before. These nodes can either have been added during different iterations, or in the same iteration since the updateParent function allows the addition of multiple nodes (with different parent nodes and different homotopy values). If more than one of these new nodes results in a homotopy value that has not been seen previously, the result would be that more than one new homotopy class is found during the same iteration.

The advantage of the modified RRT* over the standard RRT* is that the modified RRT* is able to return multiple paths, all belonging to different homotopy classes. As seen in scenarios 2 and 3, the shortest of the returned paths (i.e. the one that would have been returned by standard RRT*) is not always the best path in terms of objective function value or collision avoidance. In the second scenario, the first path leads the agent through a narrow passage as seen in Figure 5.5, and the local optimization can only make small improvements on the initial solution, which results in a trajectory with a high cost function values and a high collision risk. The results in Figure 5.7a show a collision probability of about 70 % for both iLQG and T-LQG. The second path, which is longer but avoids the narrow passage results in a trajectory with lower cost function values and much lower collision probability (about 10 % for iLQG and T-LQG as seen in Figure 5.7b). In the third scenario, shown in Figure 5.8, the first path leads the agent through an area with a higher uncertainty than the second path does. The iLQG trajectory takes a detour to lower the uncertainty, which the T-LQG trajectory fails to find, resulting in a high final covariance matrix for the T-LQG trajectory. The second path results in trajectories with lower expected cost than the first path. From the simulation results in Figure 5.9, it can be seen that the collision probability is higher for the first path for the iLQG trajectory. There is no difference in collision probability between the two paths for the T-LQG trajectory for any of the controllers. Surprisingly, the T-LQG trajectory with iLQG controller is able to bring the system closest to the goal for the first path, despite the T-LQG trajectory having higher uncertainty. The second path performs much better in terms of reaching the goal for all trajectories.

Since the shortest path will not always achieve the lowest cost function value or the lowest collision probability, it seems advantageous to use a multi-hypothesis approach where multiple paths are generated using the modified RRT*. For each of these paths an initial trajectory can be generated, and local optimization can be performed for each such trajectory. Since trajectory generation and optimization for a path computed by the modified RRT* algorithm does not depend on any other paths, these computations can be done in parallel and would not require extra computation time given hardware with sufficiently many cores.

It would also be possible to generate multiple paths from different homotopy class using other strategies as well. A naive approach would be to run the standard RRT* algorithm a number of times and compute the homotopy class for each resulting path. The shortest path for each homotopy class could then be selected. This will with high probability result in several paths that belong to the same homotopy classes. Longer paths would be less likely to be found, since RRT* optimizes the path length, which would possibly require the algorithm to be run many times in order to find the desired number of paths. Another approach would be to block paths from homotopy classes that have already been found. However, this would not work very well for this representation of homotopy classes, since the homotopy class of a path is known only once it is connected to the goal node.

6.2 Comparison of iLQG and T-LQG

From the results in Figure 5.3, Figure 5.7 and Figure 5.9 it can be seen that there is a difference in performance between the two methods when it comes to following the computed trajectories. When comparing the iLQG method to the T-LQG method, the iLQG method on average results in a final position closer to the desired final point in all scenarios, except for the first path in the third scenario (Figure 5.9a). In some of the scenarios the difference is extreme. As an example, in the first scenario, the T-LQG method only twice resulted in a distance to the goal that was smaller than 0.25, whereas the iLQG method never resulted in a distance to the goal over 0.25 (Figure 5.3). The reason behind this is most likely due to the controller and/or state estimator used by the T-LQG method rather than the level of difficulty to follow the optimized trajectories. When the iLQG method is used to compute the control policy for the trajectories generated by the T-LQG method the two methods perform more or less equally with regard to the final distance to the goal. T-LQG with iLQG controller performs better than the iLQG method on the first path of the third scenario (Figure 5.9a), and iLQG performs better on the second and third paths of the second scenario (Figures 5.7b-5.7c). For all scenarios, the T-LQG method performs better with the control policy given by the iLQG method than the control policy computed using the T-LQG method.

The two methods differ not only in how the control policy is computed, but also in how the estimated state and covariance matrix is updated during the simulation. The iLQG method uses an EKF, whereas the T-LQG method uses a different adaptation of a Kalman filter. The reason that the iLQG control policy seems to give better results could therefore be that the control policy is better, that the state estimator is better or a combination of both.

A comparison of the computation time values in Table 5.2, Table 5.4 and Table 5.6 show that the iLQG method takes a significantly longer time to run than the T-LQG method for most trajectories. T-LQG is $\mathcal{O}(ln^3)$ whereas iLQG is $\mathcal{O}(ln^6)$ where l is the planning horizon and n is the state space dimension [17], so this is

as expected. The running time for T-LQG is generally a few seconds lower when the T-LQG control policy is computed compared to when the iLQG control policy is computed. This is also as expected since the T-LQG control policy is a function only of the state estimate whereas the iLQG control policy also takes into account the covariance estimate.

A comparison of the cost function values for the two methods shows that it is difficult to draw any conclusions. For the first scenario the two methods, as seen in Table 5.1, return very similar values. The iLQG method results in a slightly lower value than the T-LQG method for the iLQG cost function, and for the T-LQG cost function the T-LQG method results in a slightly lower value. In the second scenario, as seen in Table 5.3, the T-LQG method results in lower cost function values than the iLQG method for both the iLQG cost function and the T-LQG cost function for all three paths. In the third scenario, as seen in Table 5.5, the pattern is similar to that of the first scenario, with both methods resulting in lower cost function values than the other for their respective cost function. The difference in cost function values is higher in this scenario than in the first, in particular for the first path. For the first path and the iLQG cost function, the iLQG method reduces the cost of the initial path with 75 % compared to 11 % for the T-LQG method. For the first path and the T-LQG cost function, the T-LQG method reduces the cost of the initial path with 11 %, whereas the iLQG method increases the cost with 11 %.

The main advantage of the iLQG method is that it can be implemented as an anytime algorithm. Once the control policy has been computed for the initial trajectory, the iLQG algorithm always has a feasible trajectory with a corresponding policy that it continuously improves. On the other hand, the T-LQG algorithm starts with a feasible trajectory (that may not be feasible for the NLP if the final state is not close enough to the goal state) but may relax feasibility during some intermediate iterations when trying to improve the objective function value. This trade-off between feasibility and optimality is standard for NLP solvers [11]. If the iLQG algorithm is allowed to run for a given time it will always return a feasible solution and control policy, but there is no guarantee that T-LQG will return a feasible trajectory if it is terminated after a given time.

The iLQG method shows a somewhat higher tendency to guide the solution towards areas with low uncertainty. This is demonstrated by the first path of the third scenario, seen in Figure 5.8. The iLQG trajectory (Figure 5.8c) rounds the obstacle to arrive at the area with the lowest measurement uncertainties before returning to the area with higher measurement uncertainties and reaching the goal. The T-LQG trajectory (Figure 5.8e) is shorter and does not leave the area with high measurement uncertainties. As a result the iLQG in this case has a much smaller uncertainty when arriving at the goal. This is explained by the difference in cost functions between the two methods. T-LQG has the same cost function for all time steps, but iLQG has a different cost function $c_l(\mathbf{b_l})$ for the final time step. Since the uncertainty term is weighted more in the cost function for the final time step, it will be beneficial to find trajectories that leads to an

increased value of $c_t(\mathbf{b_t})$ for some $t < l$ but decreases the final uncertainty. For the standard T-LQG cost function it may not be worth it to decrease the final uncertainty if this increases other parts of the cost function, e.g, control effort. In a modification of the third scenario, T-LQG is tweaked such that it uses different weight matrices for different time steps. The resulting trajectory, as seen in Figure 5.10, shows a similar behaviour as the iLQG trajectory, leading the agent to the area with lower measurement uncertainty. A comparison of the cost function values in Table 5.7 show that the modified T-LQG results in lower cost values for both the iLQG and T-LQG cost functions. A comparison of the computation times in Table 5.6 and Table 5.8 show only a small increase in computation time when the iLQG control policy is used, and a decrease in computation time when the T-LQG control policy is used. A comparison of Figure 5.9a and Figure 5.11 show that the final distance to the goal on average is higher when the modified cost function is used for the iLQG controller, even if it still performs better than the iLQG trajectory in terms of final distance to the goal. When the modified cost function is used, the T-LQG controller less often results in a very large (> 0.5) distance to the goal, but it also less often results in a small (≤ 0.15) distance to the goal.

7

Conclusions and future work

This chapter summarizes the main results of the thesis and presents some ideas for future work.

7.1 Conclusions

A modified version of the RRT* algorithm that returns multiple paths belonging to different homotopy classes is presented in this thesis. The algorithm has been implemented and evaluated on different scenarios. Two different methods, iLQG and T-LQG, for motion planning under uncertainty have been implemented and combined with the modified RRT*.

The two methods for motion planning under uncertainty have been evaluated in a simulation study. The results show that both methods are capable of improving an initial trajectory with regards to some cost function. The computation time for T-LQG is significantly lower than that of iLQG, due to lower time complexity. Due to differences in cost functions, it is difficult to compare their performance cost wise since each method tends to achieve the lowest cost for its own cost function. T-LQG possibly tends to result in lower cost function values overall. In one case T-LQG resulted in lower costs for both cost functions, and when the cost function of T-LQG was modified to be more similar to the iLQG function the result was that T-LQG achieved lower costs for both cost functions. A disadvantage of T-LQG is that simulations show that the collision probability is generally higher than for iLQG, and that the final distance to the goal is generally greater for T-LQG than for iLQG. If the control policy for the T-LQG trajectory is computed in the same way as for iLQG this difference in success rate becomes smaller, but computation times are slightly higher. The main advantage of iLQG is that it

can be implemented as an anytime algorithm. Another disadvantage of T-LQG is that when the modified cost function was used it failed to return a trajectory for one of the two cases it was tried on. Although the fault lies with the NLP solver rather than the T-LQG method, it indicates that T-LQG is very dependent on the choice of NLP solver, making it less robust than iLQG.

Results from scenarios with multiple trajectories (belonging to different homotopy classes) show that the multi-hypothesis approach where more than one trajectory is generated can be an advantage. The RRT* algorithm has no information about measurement uncertainties or collision risks and if only one path is returned there is a risk that the path is such that only small optimizations can be done. Examples include if the path passes through a narrow passage as in scenario 2, or if the path passes through areas of high uncertainty with small possibilities of reaching areas with low uncertainty. In these cases it is advantageous to have other paths, in other homotopy classes, that can be used to generate optimized trajectories that can be compared to each other. These computations can be implemented in parallel which would not require any additional computation time.

7.2 Future work

One possible improvement of the modified RRT* algorithm is to introduce a limit on the number of the paths it can return. Another possible modification is to change the cost function used to include measurement uncertainties and/or information about obstacles. That could give an indication of which paths the optimization should focus on.

Both iLQG and T-LQG have some aspects that it would be desirable to improve upon. The main disadvantage of iLQG is that it is is computationally expensive. There has been no attempt to optimize the implementation in this thesis, so it could be interesting to investigate how much the algorithm can be optimized or if there are parts of the algorithm that can be run in parallel. T-LQG performs worse than iLQG with respect to trajectory tracking. Future work could investigate if the problem lies with the control policy or the estimator as well as look at alternative control policies and/or estimators. The simulations of this thesis indicate that the iLQG approach to control policy improves the trajectory tracking, but it comes with the cost of increased dimensionality of the control gain matrix and increased computation time.

The simulations in this thesis have been performed on scenarios with static obstacles that do not change over time. Future work could be done on scenarios with dynamic obstacles that change position and/or extension over time.

Bibliography

[1] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019. doi: 10.1007/s12532-018-0139-4. Cited on page 33.

[2] Subhrajit Bhattacharya. Search-based path planning with homotopy class constraints. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. Cited on pages 7 and 8.

[3] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957. ISSN 00029327, 10806377. URL `http://www.jstor.org/stable/2372560`. Cited on page 8.

[4] Emili Hernández, Marc Carreras, Javier Antich, Pere Ridao, and Alberto Ortiz. A topologically guided path planner for an auv using homotopy classes. In *2011 IEEE International Conference on Robotics and Automation*, pages 2337–2343. IEEE, 2011. Cited on page 7.

[5] Ulf Janfalk. *Linjär algebra*. Matematiska institutionen Linköpings universitet, 2014. Cited on page 30.

[6] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960. Cited on page 9.

[7] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7): 846–894, 2011. Cited on page 7.

[8] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Department, Iowa State University (TR 98–11), 1998. Cited on page 7.

[9] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006. Cited on pages 1 and 38.

[10] Israel Lugo-Cárdenas, Gerardo Flores, Sergio Salazar, and Rogelio Lozano. Dubins path generation for a fixed wing UAV. In *2014 International conference on unmanned aircraft systems (ICUAS)*, pages 339–346. IEEE, 2014. Cited on page 9.

[11] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006. Cited on page 56.

[12] Nürnberg, Robert. Distance from a point to an ellipse. `http://wwwf.imperial.ac.uk/~rn/distance2ellipse.pdf`, 2006. [Online; accessed 10-March-2020]. Cited on page 31.

[13] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987. Cited on page 6.

[14] Sachin Patil, Gregory Kahn, Michael Laskey, John Schulman, Ken Goldberg, and Pieter Abbeel. Scaling up gaussian belief space planning through covariance-free trajectory optimization and automatic differentiation. In *Algorithmic foundations of robotics XI*, pages 515–533. Springer, 2015. Cited on page 11.

[15] Robert Platt Jr, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. *Proceedings of the Robotics: Science and Systems Conference*, 2010. Cited on page 11.

[16] Mohammadhussein Rafieisakhaei, Suman Chakravorty, and PR Kumar. Belief space planning simplified: Trajectory-optimized lqg (t-lqg). *arXiv preprint arXiv:1608.03013*, 2016. Cited on pages 15 and 16.

[17] Mohammadhussein Rafieisakhaei, Suman Chakravorty, and PR Kumar. T-LQG: Closed-loop belief space planning via trajectory-optimized LQG. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 649–656. IEEE, 2017. Cited on pages 11, 14, 15, 16, and 55.

[18] Steinbeis-Forschungszentrum Optimierung, Steuerung und Regelung. WORHP large-scale sparse nonlinear optimisation. `https://worhp.de`, 2018. [Online; accessed 24-March-2020]. Cited on page 34.

[19] Wen Sun, Jur van den Berg, and Ron Alterovitz. Stochastic extended lqr for optimization-based motion planning under uncertainty. *IEEE Transactions on Automation Science and Engineering*, 13(2):437–447, 2016. Cited on page 11.

[20] Sebastian Thrun, Wolfram Burgaard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005. Cited on page 5.

[21] Emanuel Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic sys-

tems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005. Cited on page 12.

[22] Jur Van Den Berg, Pieter Abbeel, and Ken Goldberg. LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913, 2011. Cited on pages 13 and 30.

[23] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012. Cited on pages 11, 12, 13, 14, and 31.

[24] Daqing Yi, Michael A Goodrich, and Kevin D Seppi. Homotopy-aware rrt*: Toward human-robot topological path-planning. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 279–286. IEEE, 2016. Cited on page 7.

[25] Huan Yu, Wenjie Lu, and Dikai Liu. A unified closed-loop motion planning approach for an i-auv in cluttered environment with localization uncertainty. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4646–4652. IEEE, 2019. Cited on page 11.